Toward Lifelong Visual Localization and Mapping

by

Hordur Johannsson

Submitted to the Dept. of Electrical Engineering and Computer science and the Joint Program in Applied Ocean Science and Engineering

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

and the

WOODS HOLE OCEANOGRAPHIC INSTITUTION

June 2013

Author .....

Joint Program in Applied Ocean Science and Engineering

February 14, 2013

Certified by .....

Professor John Leonard

Professor of Mechanical and Ocean Engineering

Thesis Supervisor

Accepted by .....

Professor Henrik Schmidt

Chair, Joint Committee for Applied Ocean Science and Engineering, WHOI

Accepted by .....

Professor Leslie A. Kolodziejski

Chair, EECS Committee on Graduate Students, MIT

Toward Lifelong Visual Localization and Mapping

by

Hordur Johannsson

Submitted to the Dept. of Electrical Engineering and Computer science and the
Joint Program in Applied Ocean Science and Engineering
on February 14, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Mobile robotic systems operating over long durations require algorithms that are robust and scale efficiently over time as sensor information is continually collected. For mobile robots one of the fundamental problems is navigation; which requires the robot to have a map of its environment, so it can plan its path and execute it. Having the robot use its perception sensors to do simultaneous localization and mapping (SLAM) is beneficial for a fully autonomous system. Extending the time horizon of operations poses problems to current SLAM algorithms, both in terms of robustness and temporal scalability. To address this problem we propose a reduced pose graph model that significantly reduces the complexity of the full pose graph model. Additionally we develop a SLAM system using two different sensor modalities: imaging sonars for underwater navigation and vision based SLAM for terrestrial applications.

Underwater navigation is one application domain that benefits from SLAM, where access to a global positioning system (GPS) is not possible. In this thesis we present SLAM systems for two underwater applications. First, we describe our implementation of real-time imaging-sonar aided navigation applied to in-situ autonomous ship hull inspection using the hovering autonomous underwater vehicle (HAUV). In addition we present an architecture that enables the fusion of information from both a sonar and a camera system. The system is evaluated using data collected during experiments on SS Curtiss and USCGC Seneca. Second, we develop a feature-based navigation system supporting multi-session mapping, and provide an algorithm for re-localizing the vehicle between missions. In addition we present a method for managing the complexity of the estimation problem as new information is received. The system is demonstrated using data collected with a REMUS vehicle equipped with

a BlueView forward-looking sonar.

The model we use for mapping builds on the pose graph representation which has been shown to be an efficient and accurate approach to SLAM. One of the problems with the pose graph formulation is that the state space continuously grows as more information is acquired. To address this problem we propose the reduced pose graph (RPG) model which partitions the space to be mapped and uses the partitions to reduce the number of poses used for estimation. To evaluate our approach, we present results using an online binocular and RGB-Depth visual SLAM system that uses place recognition both for robustness and multi-session operation. Additionally, to enable large-scale indoor mapping, our system automatically detects elevator rides based on accelerometer data. We demonstrate long-term mapping using approximately nine hours of data collected in the MIT Stata Center over the course of six months. Ground truth, derived by aligning laser scans to existing floor plans, is used to evaluate the global accuracy of the system. Our results illustrate the capability of our visual SLAM system to map a large scale environment over an extended period of time.

Thesis Supervisor: Professor John Leonard
Title: Professor of Mechanical and Ocean Engineering

Acknowledgments

My time here in the MIT/WHOI program has been a wonderful experience both academically and personally. To get the chance to deeply explore topics in my field has been a real privilege. Along the way I have made many friends and found intelligent and interesting people which I'm truly thankful to have met.

First, I'm really grateful to my advisor, John Leonard, for his support and encouragement through the years. He has truly made this time a great time, giving me the opportunity to work on so many exciting projects over the years. I would also like to thank Marby Darbey for her helpful comments on the writing.

Next, I would like to thank my committee: Daniela Rus, Hanumant Singh, Michael Kaess, and Seth Teller. They have provided me with valuable guidance along the way. I really enjoyed my time in Hanu's lab during the times I spent in Woods Hole.

Also, I would like to thank the people I have met and worked with here at MIT. It has been a privilege working with Michael Kaess and Maurice Fallon on so many projects over the years, enjoying random discussion, coffee and sharing ideas on our research. I'm really thankful to all the great people in my lab, Michael Benjamin, Rob Truax, Aisha Walcott, David Rosen, Mark VanMiddlesworth, Ross Finman, Elias Mueggler, and Dehann Fourie. They really made this a fun time.

I've also been lucky enough to collaborate with many amazing people outside my group: Abe and Daniel Maturana on the Visual SLAM, and on the HAUV project I had great times working with Brendan Englot and Franz Hover at MIT, and Ryan Eustice and Ayoung Kim from the University of Michigan. On the HAUV project it has been a pleasure working with the people at Bluefin Robotics, especially Jerome Vaganay, Kim Shurn and Mike Elkins; their preparedness and quick problem solving

skills have made our operations together successful.

Finally, this journey would only have been half as bright if not for my family. I'm really thankful to my mom for her love and support, for her visiting so often to spend time with us here in Cambridge. I would like to thank my daughter Bjarney, for her inspiration and smiles, and, to my wife, Gyda, I'm forever grateful for your love and care every step of the way.

Thank you,

Hordur

This work was funded by the Office of Naval Research under grants N00014-05-1-0244, N00014-11-1-0119, N00014-06-1-0043, N00014-10-1-0936 N00014-11-1-0688, and N00014-12-1-0020.

Contents

List of Figures

List of Tables

List of Acronyms

PG pose graph

RPG reduced pose graph

GPS global positioning system

DVL Doppler velocity log

DIDSON dual frequency identification sonar

GPU graphical processing unit

LBL long-baseline acoustic navigation

RLG ring laser gyro

FOG fiber optic gyro

AUV autonomous underwater vehicle

HAUV hovering autonomous underwater vehicle

ESEIF exactly sparse extended information filter

SLAM simultaneous localization and mapping

EKF extended Kalman filter

EIF extended information filter

SEIF sparse extended information filter

ESDF exactly sparse delayed-state filter

SAM smoothing and mapping

iSAM incremental smoothing and mapping

IMU inertial measurement unit

USBL ultra short base line

UKF unscented Kalman filter

NDT normal distribution transform

ROV remotely operated vehicle

ML Maximum Likelihood

VO visual odometry

FOVIS Fast Odometry for VISion

LM LevenbergMarquardt

SAD sum of absolute differences

OOI ocean observatories initiative

LCM lightweight communication and marshaling

WO wheel odometry

Chapter 1

Introduction

Despite substantial progress in robotic mapping in the past two decades, many challenges remain before we can create truly autonomous mobile robots that can safely and reliably navigate for long durations in dynamic environments. This thesis investigates the problems of persistence and robustness for robotic mapping, presenting new algorithms that address temporal scalability in mapping. These approaches are validated using extensive implementations with real data acquired by a variety of robotic platforms, operating in both terrestrial and undersea environments.

Long-term deployment of autonomous vehicles has advantages for both scientific and commercial applications. When studying the oceans there is often interest in the study of processes such as nutrient and carbon cycles, and bio-diversity, over long time-scales, which has initiated work towards long term deployments of underwater vehicles [33]. Another area of interest would be continuous mapping of marine formations like hydrothermal vents [113]. Routine inspection of marine structures, such as ship hulls, pier pilings, and pipelines, is another application for long-term or repeated deployments of AUV's.

However, this poses many challenges in terms of power systems, mechanical design, navigation, and autonomy. In particular, when operating in complex environments or near the sea floor, precise navigation is of vital importance. In this thesis we will focus on navigation for both underwater and ground vehicles, and in particular on mapping and localization for long durations which we refer to as lifelong mapping.

1.1 Challenges in Localization and Mapping

To achieve long-term robotic autonomy, in complex and dynamic environments, mapping algorithms which scale solely with the area explored and not in exploration time are required. When operating in an unknown environment the robot needs to concurrently construct a map and localize — this is known as simultaneous localization and mapping (SLAM). In recent work, Lim et al. [67] report on a state of the art large-scale visual SLAM system. In conclusion they write:

> One interesting future direction is to keep the number of keyframes and landmarks to a manageable size by merging or trimming redundant map entries when the robot stays in or visits the same place many times. This would allow the map size to be proportional to the size of the space and not to the time spent in the space. [67]

This is the key problem addressed by this thesis. Additionally, long-term persistent operation will require the ability to develop compact representations, which can effectively describe an environment of interest, yet still provide robustness to changes in the environment and recovery from mistakes. Furthermore, when repeatedly surveying the same area we want to localize the vehicle with maps constructed from

previous operations and eventually combining them into a new up-to-date description of the environment.

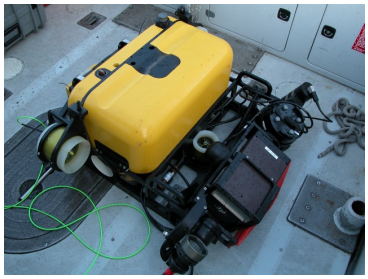
Many recent solutions to mapping are based on the pose graph formulation [68]. In this formulation the world is represented by a set of discrete poses sampled along the full trajectory of the robot, which are connected by odometry and loop closure constraints. Very efficient recursive algorithms have been presented which can maintain an online solution to this continuously expanding optimization problem. However the pose graph, by design, grows unbounded in time as recognized by [2]. This is true even for small environments which are repeatedly explored, making the use of the full pose graph unsuitable for long-term mapping.

1.2 Applications of Long-term Mapping

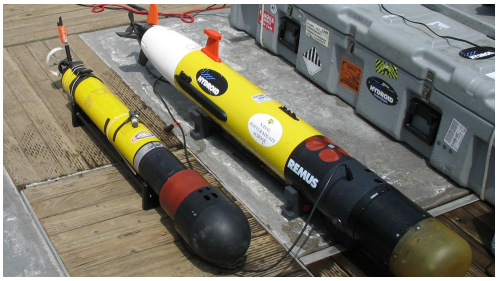
Using perception sensors for autonomous localization and mapping is beneficial for many applications. In both indoor and underwater domains it is often difficult to obtain access to external positioning information such as global positioning system (GPS). Even though underwater long-baseline acoustic navigation (LBL) beacons are often deployed it is still desirable to operate without any external infrastructure.

Ship Hull Inspection

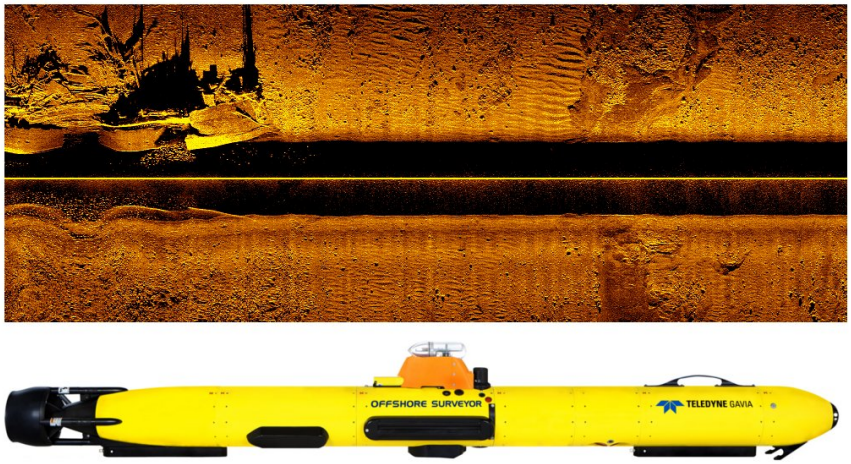
To ensure safety around their vessels the U.S. Navy routinely carries out a detailed ship hull inspection on their ships. In recent years there has been extensive development of technologies to automate this task [42] — which previously has been performed by divers and marine mammals. Automating this task is desirable both from a safety perspective and also to enable new capabilities such as accurate localization of targets, navigation back to targets for close inspection, verifying object



(a) HAUV - ship hull inspection vehicle



(b) Remus - survey vehicle



(c) Gavia AUV and an example side scan sonar image from the vehicle.

(d) PR2 - an indoor service robot

(e) Man portable mapping unit

Figure 1-1: Various mapping platforms

removal, and detection of change over time as multiple missions are performed on the same vessel. The vehicle that was developed for the ship hull inspection is the hovering autonomous underwater vehicle (HAUV), shown in Figure 1-1(a).

Seafloor Surveying

Another application domain is seafloor surveying. For example Folkesson et al. [31] used SLAM on a small vehicle to reacquire targets using a prior map, which is created using a REMUS autonomous underwater vehicle (AUV) (see Figure 1-1(b)) equipped with good navigation sensors and a side scan sonar. An exciting project for lifelong autonomy, that will soon go online, is the ocean observatories initiative (OOI) [33]. The OOI consists of a cyberinfrastructure to allow distributed observations of coastal and deep ocean environments. A part of that system is the Pioneer array which consists of several AUVs and gliders. An extension to this observatory could be AUVs equipped with imaging sensors to continuously survey the seafloor around the sites. In this case having a lifelong SLAM system would be essential. Survey vehicles would maintain an accurate map of the environment. The maps could then be shared with inspection vehicles that would use them to safely navigate towards objects of interest for further investigation.

Service Robots

Robots are useful in many other application domains and have been successfully deployed in hospitals to help with various delivery tasks for items such as linens and medicine. To navigate successfully they are provided with floor plans or manually driven around to construct an initial map. Also RFID tags can be deployed to provide fixed references [4]. The Willow Garage's PR2 robot, shown in Figure 1-1(d), has

successfully operated over extended periods of time using a prior occupancy grid map, using a laser scanner for localization and obstacle avoidance [71]. However, it would be beneficial to fully autonomously construct and maintain those maps.

Man-portable Mapping

Even though our main focus is on SLAM in the context of robotics it is also worth mentioning that the sensors are not required to be carried by a robot to be useful [25]. For example, first responders could be tracked as they enter a collapsed building, and even though a map existed, the environment might have changed after catastrophic events like earthquakes or explosions. The responder could be equipped with a sensor rig as shown in Figure 1-1(e) that would be used to construct a new map of the environment. Having this map would be useful for keeping track of which areas had been inspected, knowing the location of the person, and improved situational awareness for the person carrying the mapping gear.

1.3 Mapping Approach

The two primary services of any mapping system are: 1) provide a map that can be used for planning, 2) estimate a location and orientation relative to the map so the planned path can be executed. When constructing the map one should choose an appropriate representation that will depend on the available sensors, the environment and the requirements of the application.

Representation

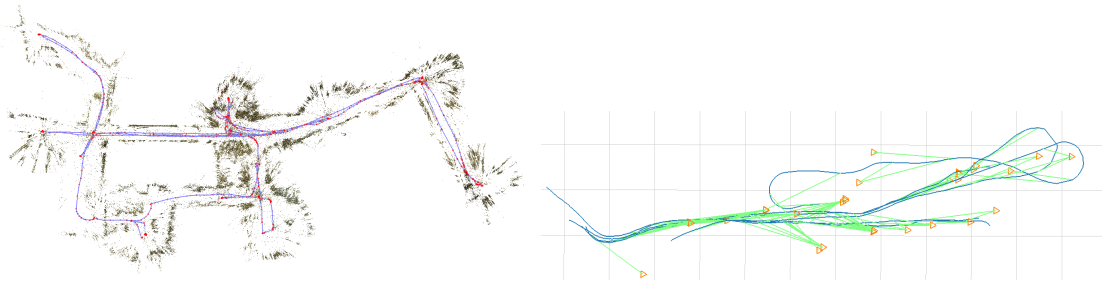
There are many possible representations for the map: feature based maps [94], pose graphs [68], poses and features, 2D and 3D occupancy grids [97,111], and topological

graphs [59]. Occupancy grids have been particularly successful for localization and path planning. Using a graph representation does not exclude their use because they can be constructed from the information stored in the pose graphs. There are metric and relative representations, and hybrid approaches [7,67]. In addition to the core mapping service it can be useful to allow other algorithms, e.g. object detection and semantic inference, to associate their results to the map. Here the graph-based representations are useful because the information can be associated to individual nodes in the graph so their locations get updated as the map estimation proceeds.

While a topological representation would work well in an environment consisting of constrained passages, a metric representation would be a better choice when operating in an open field with sparse features, as mentioned earlier in the ship hull application. A metric pose graph implicitly includes the topological graph of the environment, so any benefits from the graph-based path planners is not lost.

The type of sensor used will also affect the mapping, and the choice of sensor will depend on the application. When operating underwater, sonars are often used because of their range and ability to sense in turbid waters. For terrestrial application laser range finders have been successfully used though they typically require planar motion when used for SLAM. Cameras are an excellent sensor for localization and mapping in that they are relatively inexpensive and give rich information about the environment, facilitating tasks such as recognizing different places and detecting objects.

In our work we have chosen to use metrically accurate representations, based on a pose graph formulation both with and without features (see Figure 1.3). Metric representations fit well with our inspection and target re-acquisition applications. We will present mapping solutions using both sonars and cameras. In our case we consider only cameras that provide depth information, i.e. stereo or RGB-D cam-



(a) Stereo pose graph (b) Feature based map

Figure 1-2: Examples of two maps used in our work. On the left is a pose graph constructed using a stereo camera. To the right is a feature-based map constructed using an imaging-sonar on an underwater vehicle.

eras. In addition we also incorporate measurements from other available sensors, e.g. Doppler velocity log (DVL), pressure and inertial measurement unit (IMU) in case of the AUVs and wheel odometry and IMU for our indoor application.

Temporal Scalability

Our long-term mapping approach, called the reduced pose graph (RPG), addresses the temporal scalability of traditional pose graphs. For long-term mapping, the size of the optimization problem should be bounded by the area of the explored environment and be independent of the operation time. The reduced pose graph reuses already existing poses in previously mapped areas, which reduced the number of poses added to map. A key insight is that new measurements can still be used to further improve the map, by converting them into constraints between existing poses. The process is fluid, with new poses being added when new spaces are explored. We also discuss how the algorithm can be extended to eventually bound the time dependent complexity of the reduced pose graph.

The advantages of the reduced pose graph extend beyond scalability. Our ap-

proach maintains multiple constraints between each pair of poses. While these constraints could be combined immediately, retaining redundancy allows for consistency checking and the detection of faulty constraints. In combination with a robust estimator this can limit the effect of erroneous constraints on the state estimation. When consensus is reached over a significant number of constraints, they can eventually be combined into a single constraint, avoiding the incorporation of bad constraints into the combined edge.

Evaluation

To evaluate the reduced pose graph we developed a full 6DoF visual SLAM system. The system uses either stereo or RGB-D cameras, operates in real-time, and has been tested with data from multiple robotic platforms. A visual odometry model produces incremental constraints between key-frames which are used as input to the reduced pose graph. A place recognition module uses appearance-based methods to propose loop closures for which a geometric consistency check provides the actual constraint if successful. Place recognition allows mapping over multiple sessions and allows re-localization in case of localization failure.

Robustness can be further improved by using other sources of egomotion. In our work we have utilized wheel odometry and an IMU. An accelerometer is particularly useful to eliminate drift in inclination, which can accumulate in explorations as large as those presented here.

We have evaluated our approach on data recorded with a PR2 mobile robot from Willow Garage equipped with a stereo and an RGB-D camera which are both supported by our mapping system. Nine hours of data were used, corresponding to eleven kilometers of robot trajectory recorded over a six month period. To allow

for operation in large multi-floor indoor environments, our SLAM system can automatically detect elevator transitions. Using an accelerometer, this approach detects characteristic elevator motion to track the vertical displacement of the robot.

1.4 Contributions

This thesis makes the following contributions:

1. We present an imaging-sonar based SLAM system that was applied to autonomous ship hull inspection [42, 46].

2. We describe a multi-session feature based SLAM system for an underwater vehicle equipped with a forward looking sonar.

3. We develop a complete pose graph based visual SLAM system [47].

4. To manage the time complexity of the pose graph we propose and evaluate a reduced pose graph model [47].

5. Finally, we present an efficient GPU based scene prediction algorithm that was applied to localization using an RGB-D camera [25].

1.5 Overview

In this section we give an overview of the thesis.

Chapter 2 defines the SLAM problem and reviews prior work in that area. We introduce notation and other preliminaries required for the rest of the thesis.

Chapter 3 develops an imaging-sonar aided navigation algorithm that is applied to ship hull inspection. In addition, multi-session feature-based navigation for AUVs is described, using a feature-based pose graph map representation.

Chapter 4 introduces a visual SLAM system that uses a pose graph representation and fuses additional sensor input from wheel odometry and IMU sensors — allowing mapping of large spaces in a multi-floor building.

Chapter 5 extends the visual SLAM system, introduced in the previous chapter, to address temporal scalability when operating repeatedly in the same environment, using a reduced pose graph representation.

Chapter 6 describes a localization application of the visual SLAM system when RGB-D cameras are available. This is achieved using an efficient graphical processing unit (GPU) based depth image prediction algorithm.

Chapter 7 concludes the thesis with a review of the main contributions of the work presented in the thesis and a discussion of future research topics.

Chapter 2

SLAM Literature Review

In this chapter we will give an overview of major developments on the SLAM problem and in particular review work in the area of lifelong operations. We will review the theoretical background of the SLAM problem and introduce notation used in the rest of the thesis.

2.1 Overview of SLAM

Some of the early work on SLAM includes the seminal paper by Smith et al. [94]. They posed the problem in a probabilistic framework by using measurements of spatial relations and proposed using an Extended Kalman Filter (EKF) to estimate the first and second moments of the probability distribution for the relations. This method was used with landmarks and sonar range measurements by Leonard et al. [66]. They noted that the size of the state vector would need to grow linearly with the number of landmarks and it was necessary to maintain the full correlation between all the variables being estimated, thus the algorithm scales quadratically,

$O(N^2)$, with the number of variables, N, in time and space.

The Sparse Extended Information Filter (SEIF) was developed by Thrun et al. [100] to address these scalability issues. It used the information form of the EKF in combination with a sparsification method. A drawback of that approach was that it produced over-confident estimates. That was addressed in the exactly sparse delayed-state filters (ESDFs) by Eustice et al. [20, 22] and later with the Exactly Sparse Extended Information Filter (ESEIF) by Walter et al. [105].

Particle filters have also been used to address the complexity and data association problem. The estimates of the landmark locations become independent when conditioned on the vehicle trajectory. This fact was used by Montemerlo et al. [75] to implement FastSLAM. The main drawback with the particle filters is that there is always the risk of particle depletion. This is especially the case in the SLAM problem which is very high dimensional. Monte Carlo localization (MCL), which uses a particle filter, has also been successful given static prior maps and was used by the museum guide Minerva [98]. More recently MCL has been used with a RGB-D camera to localize in a complex 3-D environment [26].

One of the problems with the filtering approaches is that at each step the measurements are linearized around the current estimate. This can cause inconsistency of the estimate [48]. Also, before a loop closure is applied the current estimate can be far from the true value. It is also difficult to apply delayed measurements or revert a measurement that has been applied to the filter. The Atlas framework by Bosse et al. [5] addresses these issues by combining local sub-maps and a nonlinear optimization to globally align the sub-maps. Each sub-map has its own local coordinate so the linearization point will not deviate as far from the true value as it does in the case of global parametrization.

The pose graph optimization approach to SLAM was first introduced by Lu and

Milios [68] and further developed by many researchers including Gutmann and Konolige [37], Folkesson and Christensen [28], Dellaert [14] and Olson et al. [82]. Significant research has focused on providing efficient solutions, both approximate and exact. There have been many recent extensions to pose graph SLAM to address optimization efficiency. Notable recent examples include: hierarchical representations (Grisetti et al. [35]), collection of local maps (Bosse et al. [6]; Estrado et al. [19]; Ni et al. [78]) as well as relative and non-Euclidean (Newman et al. [77]; Sibley et al. [92]) approaches. However few have addressed reducing the growth in size of the number of pose graph nodes as a function of time.

The SLAM problem can be modeled as a graph where the nodes represent the vehicle poses and landmarks. The edges are constraints on these variables and are derived from sensor measurements. Often only the poses are estimated and not the landmarks explicitly. By associating probability distributions to the constraints the graph can be interpreted as a Bayes network. If it is assumed the measurements are corrupted by zero-mean Gaussian noise then the maximum likelihood solution of the joint probability distribution can be found by solving a nonlinear least squares problem. That is the approach Dellaert et al. used in the square root smoothing and mapping algorithm [14, 15]. It used Cholesky factorization to efficiently solve the normal equations of the sparse least-squares problem. Furthermore, they related the graphical model to the matrices associated with the linearized optimization problem. The downside is that the problem grows with time instead of the size of the environment. How to extend this to cases of lifelong mapping is one of the problems investigated in this thesis.

An extension to this method is the incremental Smoothing and Mapping (iSAM) that was developed by Kaess et al. [53]. In iSAM the Cholesky factorization is incrementally updated using Givens rotations, making the method better for on-

line operations. In addition they developed an algorithm for recovering parts of the covariance matrix, which is useful for on-line data association decisions. Working directly with the graphical model, the Bayes-Tree [51] algorithm was developed and used in iSAM 2.0 [52]. The algorithm works by constructing and maintaining a junction-tree which enables implementation of fluid re-linearization and incremental variable re-ordering.

Another incremental method is HoG-Man [35], which trades accuracy for speed by creating a hierarchical approximation to the full estimation problem. These methods belong to the class of direct solvers. Another class of algorithms are iterative solvers like the stochastic gradient descent which was used in Olson's work [82] and Grisetti's TORO [36].

Recently Sibley et al. [91] and Mei et al. [72] have proposed using a relative parametrization for visual SLAM. They argue that a locally accurate map is sufficient to find the shortest paths between places. The benefit of the relative parametrization is that the updates are local even for large areas as they have demonstrated in their work.

Many sensors have been used for the SLAM problem, including: sonars, laser range finders and cameras. With increasing computation power and the availability of inexpensive cameras, using vision in SLAM has become increasingly popular. Both monocular [13,57] and stereo vision [60] have been used to estimate egomotion. One of the more difficult challenges in SLAM is recognizing when a place is revisited. Inspired by text-based search engines like Google, Sivic et al. [93] proposed learning visual words and then using inverted indexes, as is used for document retrieval, to search for similar images. This was later used by Cummins et al. [12] in FAB-MAP, which uses the visual appearance of places to localize and construct a topological map of the world. Other similar approaches are the vocabulary tree by Nister et

al. [80] which was used in Konolige's visual maps [61].

Also, more recently inexpensive RGB-Depth cameras have become available. The RGB-D camera poses interesting new questions, as it is both a color camera and a depth sensor based on stereo vision. Because it is an active system, its range is limited compared to that of a regular stereo camera. These two sensor modalities, the sparse features and dense point clouds, can complement each other [41], and considering different ways to use this large amount of 3D data is an interesting research problem.

2.2 Underwater navigation

Navigation is a challenging and important capability for underwater vehicles. Kinsey et al. provide a comprehensive survey of available AUV navigation sensing options [56]. Pressure sensors are frequently used to measure the depth of the vehicle from the surface. The vehicle speed can be estimated with a motion model and a propeller count. For more accurate navigation Doppler Velocity Logs (DVLs) are used, which provide the vehicle velocity relative to a surface, and also measure velocity relative to the water. For orientation, magnetic compasses are used as well as gyroscopes, including fiber optic gyro (FOG) and ring laser gyro (RLG). Inertial measurement units (IMU) measure acceleration and can further improve navigational accuracy, giving a measure of the gravity vector. Finally, for high-precision surveying these sensors are often combined into an inertial navigation system (INS) providing accuracy up to 0.05% of distance traveled [83].

The sensors mentioned above provide basic dead reckoning for the vehicle, but the position estimate will drift over time. To bound the navigational drift, time of flight measurements with acoustic beacons are commonly used in underwater navigation [108, 109, 114]. Long baseline (LBL) and ultra-short baseline (USBL) sytems have

proven successful in various applications, such as underwater archeology [74] and ship hull inspection [39].

In some situations, e.g. when there is no direct line of sight, or when the environment causes multi-path returns, it can be beneficial to use SLAM with the vehicle perception sensors to bound the drift of the vehicle navigation instead of deploying acoustic beacons. This saves time by not having to deploy and calibrate the beacons. There have been various approaches based on using both sonars and cameras. Eustice et al. [21] proposed a view-based approach using constraints from overlapping camera frames using an information filter to estimate vehicle pose. Building bathymetry submaps and aligning them together in an EKF framework was used by Roman et al. [88]. Ribas et al. [87] and Mallios et al. [69] mapped structured environments using an Extended Kalman filter with scanning sonars and a scan matching alignment. A particle filter for a 3D occupancy grid was used by Fairfield et al. to navigate a cave inspection vehicle [23]. In work by Folkesson et al. [30], a forward-looking sonar was used with a prior map to track features in sonar images and use them to localize the vehicle. Imaging sonars have also been used as navigation aids in ship-hull inspection using either filtering [18, 104] or smoothing [46] for the map estimation. Recently Kunz used a factor graph representation and a nonlinear solver to perform accurate mapping of moving ice floes using cameras and multibeam sonars [65].

2.3 Approaches to Lifelong Mapping

Some of the early work on lifelong SLAM is the adaptive place network by Yamauchi et al. [112], which is a topological roadmap that updates confidence on links as the robot traverses them; learning environmental configuration using lasers scans by Stachniss et al. [95]; and tracking samples from the environment at different

timescales by Biber et al. [1, 2]. An initial map is created with traditional SLAM methods, and then updated at multiple different time scales to capture dynamic changes in the environment. The map is represented by a set of evolving grid-based local maps connected to an underlying pose graph. They demonstrate long-term mapping using several hours of data recorded over five weeks. This work is specific to laser-range data.

The ATLAS framework by Bosse et al. [6] made important contributions in graph formulation, robustness by cycle verification and uncertainty estimation using Dijkstra projection, although it did not address lifelong mapping in particular. A relative formulation is used, though it is pointed out that a global estimate can be achieved by finding a global configuration of all the submaps that minimize the error of edges between submaps.

Compact pose SLAM by Ila et al. [45] uses an information-theoretic method to decide which constraints should be added. New poses are added to the estimator (an information filter) only if no other poses are nearby, while taking into account information gain from potential loop closures. The paper does not address how to limit growth when continuously operating in the same environment. In contrast, our approach can connect constraints to existing poses in areas already mapped — so as to avoid the need for the periodic addition of new nodes along the trajectory.

Kretzschmar et al. [63, 96] also use an information-theoretic approach to decide which laser scan should be removed from the graph. They have shown large reductions in complexity for laser-based pose graphs, using an approximate marginalization to retain the sparsity of the solution.

An algorithm for complexity reduction was proposed by Eade et al. [17]. The reduction consists of marginalization and degree thresholding. When the degree of a node exceeds a given threshold the constraint with the least residual error is removed.

This heuristic will cause a bias in the estimate, e.g. for repeated passes through the environment the measurements that survive are those with the largest residual error.

The visual maps by Konolige et al. [62] are closely related to our work. They create a skeleton graph of views similar to a pose graph. To keep the density of views or poses constant in a given region, least-recently used views are removed from the skeleton by marginalization. Our work, in contrast, partitions the environment and avoids adding redundant views to begin with. Other related recent work in multi-session visual SLAM was presented by McDonald et al. [70], which combines multiple mapping sessions in a pose graph optimization framework, with appearance-based loop closing [9], however this work did not address temporal scalability.

The dynamic pose graph [103] explicitly looks for changes in the environment and removes nodes where the environment has changed. The main limitation of this approach is that the temporal scalability will depend on how rapidly the environment changes. Instead we conjecture that it is sufficient to track an active set of poses that covers the environment. Where given a partitioning of the space, and each pose sees some of the partitions, then the cover is a set of poses that see all the partitions.

For monocular SLAM, a different approach without pose graphs has been taken for managing complexity when repeatedly mapping the same environment. Most notably, Klein et al. [57, 58] introduced monocular parallel tracking and mapping (PTAM), where a map of sparse features is updated over time by bundle adjustment, and the camera is continuously localized based on this map. Targeting augmented reality applications, the original formulation of PTAM was limited to small scale environments, mostly because of the complexity of bundle adjustment.

An extension to augmented reality applications to large-scale environments using several local PTAM maps is demonstrated by Castel et al. [10]. More recently, Pirker et al. [85] proposed larger scale monocular reconstruction, again based on bundle

adjustment. Their system updates the map in dynamic environments and achieves real-time performance with the exception of loop closures.

A completely different approach to long-term mapping is taken by Milford et al. [73] in biologically inspired work. Their RatSLAM system used panoramic video and odometry as perceptual stimuli. While their work does not try to produce Cartesian maps, they have demonstrated impressive long-term mapping and navigation in dynamic environments.

2.4 Simultaneous Localization and Mapping

The SLAM problem can be formulated in the following way. Let $X = \{x_i\}$ be the robot's pose at discrete time steps $i = 1, \cdots, N$. The robot has a motion model given controls $U = \{u_i\}$. The motion model can be described as a function from the previous state and given controls to a new state. Because of sensor noise and uncertainties in the motion model it is common to model the process noise as zero-mean additive noise as in the following equation

$$x_i = f(x_{i-1}, u_{i-1}) + w_i \qquad w_i \sim \mathcal{N}(0, \Sigma_i) \qquad \text{for } i > 0 \tag{2.1}$$

with an initial condition for x_0.

If the robot uses only this information then it is simply doing dead-reckoning (DR) and the position error relative to past states will grow unbounded. This is the core of the SLAM problem. The robot needs to use some of its sensor measurements to recognize that it has arrived at a previously visited place and compute its position

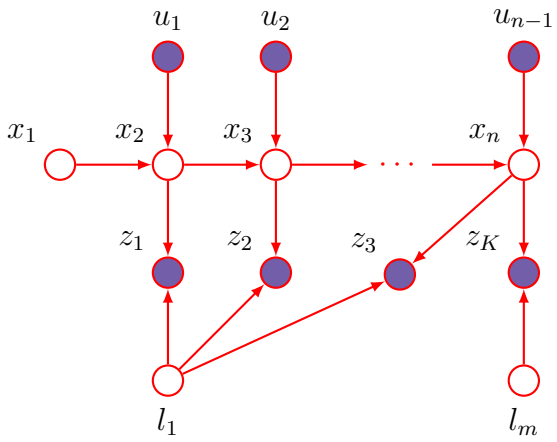


Figure 2-1: Graphical model for feature based SLAM for N poses, M landmarks, and K landmark measurements.

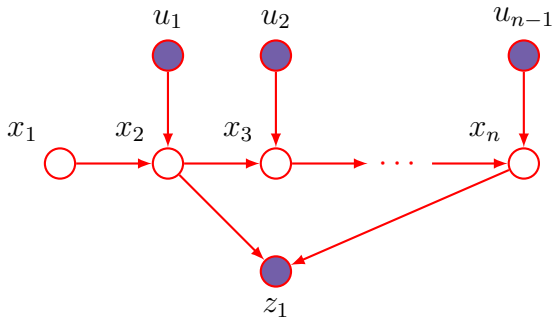


Figure 2-2: Graphical model for a pose graph.

relative to that place. This adds a sensor measurement to the formulation

$$z_k = h(x_i, x_j) + v_k \qquad\qquad v_k \sim \mathcal{N}(0, \Lambda_k) \tag{2.2}$$

where z_k is the k-th measurement. This is commonly referred to as pose graphs [82]. It is also possible to explicitly include landmarks in the state estimation as shown in Figure 2-1. Then the measurement function changes to

$$z_k = h(x_i, l_j) + v_k \qquad\qquad v_k \sim \mathcal{N}(0, \Lambda_k) \tag{2.3}$$

A natural solution is to consider the joint probability distribution over all the variables, which is given by the probability density function $p(X, Z, U)$ and is pictured as the graphical model in Figure 2-2 for the pose graph or in the landmark case $p(X, Z, L, U)$.

The graphical model for the pose graph is equivalent to:

$$p(X, Z, U) = p(x_1) \prod_{i=2}^{N} p(x_i | x_{i-1}, u_{i-1}) \prod_{(i,j)} p(z_k | x_i, x_j) \tag{2.4}$$

The graphical model shows the common assumption of a Markovian motion model which makes it feasible to solve the problem efficiently.

2.4.1 Filtering

The map can be estimated by filtering. In the landmark case one is interested in computing the posterior $p(x_t, l_{1:k_t} | z_{1:t}, u_{1:t})$ which can be computed recursively [99]

$$p(x_t, l_{1:k_t} | z_{1:t}, u_{1:t}) = \frac{p(z_t | x_t)}{p(x_t)} \int p(x_t | u_t, x_{t-1}) p(x_{t-1}, l_{1:k_{t-1}} | z_{1:t-1}, u_{1:t-1}) \mathrm{d}x_{t-1} \tag{2.5}$$

If the errors are normally distributed and the functions linear, then this distribution is a multivariate Gaussian distribution and can be solved recursively using a Kalman Filter (KF). If the functions are nonlinear, then an Extended Kalman Filter (EKF) can be used by linearizing around the current estimate, which is an approximation to the full filter presented above. A recursive solution is of particular interest for robotics applications because the data are acquired on-line as the vehicle navigates around the environment.

The filtering approach poses several challenges of computational complexity and accuracy as more information is acquired. As previously mentioned, some of the computational complexity challenges have been addressed with the information form of the EKF. One limitation that relates to robustness is that when using filtering there is no easy way to remove previous measurements if it is discovered that a measurement is incorrect. Additionally the linearization point cannot be changed at a later time. This is an issue if the mean changes drastically when loop closures are later applied.

2.4.2 Smoothing

Recently there have been many advancements using smoothing of the full trajectory for the SLAM problem, which is the approach taken in this thesis. Smoothing, in contrast to filtering, solves for all the poses and repeatedly re-linearizes the problem as the trajectory estimate is updated. Smoothing has several advantages over filtering approaches. First, data association does not need to be fixed for all time. If, at a later time, the data association decision is determined to be incorrect, it is easy to recover from that by simply removing the measurement. Secondly, when measurements are applied they do not need to be fixed at a single linearization point but can be

adjusted as the estimate improves. Finally, in many cases solving the full problem can be computationally more efficient because it retains the sparsity of the problem.

When smoothing we are interested in the full posterior $p(X|U,Z)$. One solution is to compute the maximum likelihood estimator X^* by

$$X^* = \underset{X}{\operatorname{argmax}}\, p(X|Z) = \underset{X}{\operatorname{argmax}} \prod_{k=1}^{M} p(z_k|X_k) \tag{2.6}$$

$$\tag{2.7}$$

$$= \underset{X}{\operatorname{argmin}} \sum_{k=1}^{M} -\ln p(z_k|X_k) \tag{2.8}$$

and for distributions of the form $p(z_k|X_k) \propto e^{C_k(f(X_k)-z_k)}$ we get the following non-linear optimization problem

$$X^* = \underset{X}{\operatorname{argmin}} \sum_{k} C_k \left(f(x_{i_k}, x_{j_k}) - z_k\right), \tag{2.9}$$

where f is a function that predicts a constraint from two poses and C_k is a cost function associated with constraint k. The same solution applies to the traditional pose graph with the only difference being the way that constraints and nodes are created. For a Gaussian error distribution the cost function is the Mahalanobis distance.

Another convenient graphical representation for the model are factor graphs [64], an example of which is pictured in 2-3. A factor graph is a bipartite graph $G = (\mathcal{F}, \mathcal{X}, \epsilon)$ with two node types: factor nodes $f_i \in \mathcal{F}$ and variable nodes $x_j \in \mathcal{X}$. Edges $e_{ij}\epsilon$ are always between factor nodes and variable nodes. The factor graph

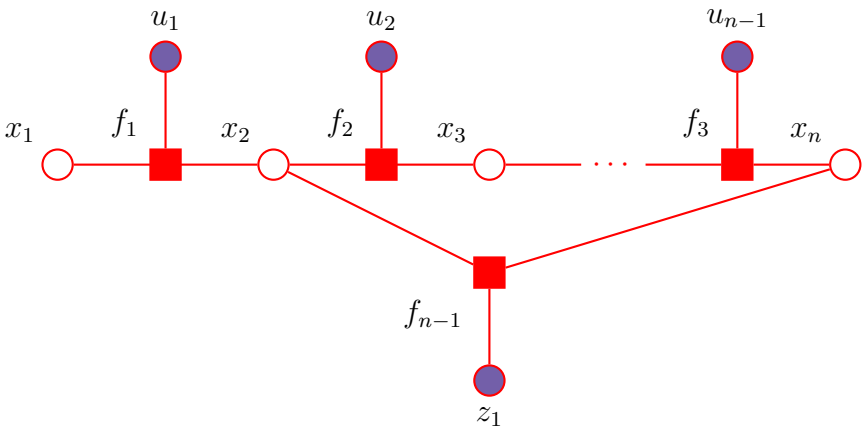


Figure 2-3: A factor graph equivalent to the Bayes network in Figure 2-2.

defines a factorization of a function fX as

$$f(X) = \prod_i f_i(X_i) \tag{2.10}$$

where X_i is the set of variables that connect to f_i. Looking at the factorization we presented earlier for the posterior $p(X|U, Z)$ we see that the measurement models $p(z|X_k)$ map directly to factors. One of the benefits of this is that it is easy to make the factorization explicit and it is close to the actual implementation of the optimization algorithm.

In this chapter we have provided an overview of SLAM. In addition we described the pose graph formulation and the pose graph with landmarks. In the following chapters we will use these models for localization and mapping using both imaging-sonars and cameras. Furthermore, we propose the reduced pose graph to address the temporal scalability issues of the full pose graph formulation.

Chapter 3

Multi-session Feature-based Navigation for AUVs

The oceans cover a large portion of our planet; they are rich in resources, important for transportation, and play a major role in Earth's climate. Thus underwater operations have gained much interest by several groups, including oceanographers, oil-industry, and the world's navies. The use of autonomous underwater vehicles (AUVs) is playing larger and larger role in many areas such as under-ice operations, mine-countermeasures, harbor surveillance, and pipeline inspection, to name a few.

For underwater vehicles, accurate navigation is important for avoiding any hazards in the environment and can save operation time by enabling precise execution of the planned trajectory. This has resulted in development of accurate navigation sensors like, Doppler velocity log (DVL) for velocity, and a fiber optic gyro (FOG) for accurate orientation. In addition external acoustic beacons like long-baseline (LBL) and ultra-short baseline (USBL) are a common option to provide accurate navigation and correct for long-term drift. Still it is desirable to have accurate navigation

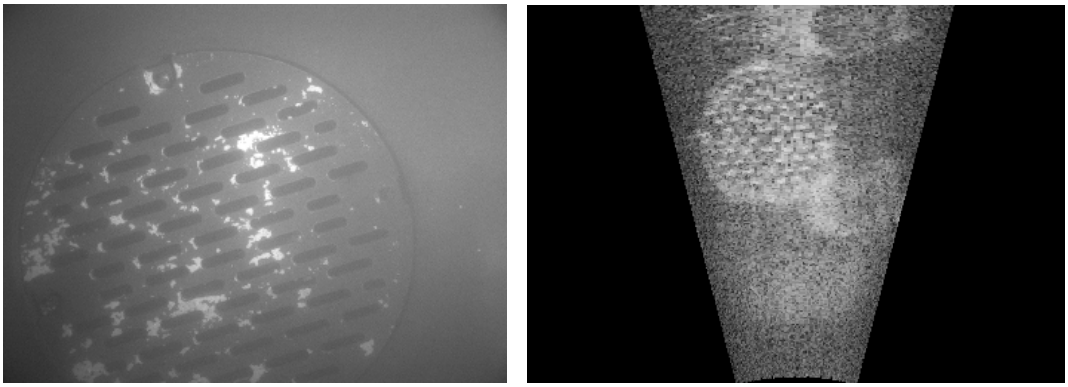
(a) HAUV - Ship hull inspection vehicle

(b) REMUS 100 - Survey vehicle

Figure 3-1: The two vehicles used for the experiments described in this chapter. The HAUV, shown on the left, was used for the experiments that will be presented in Section 3.2. The REMUS vehicle, shown on the right, was used for the experiments that will be presented in Section 3.3.

without the support from any external infrastructure.

In this chapter we will explore two methods that use a sonar system to aid the vehicle navigation system. First, we will present an imaging-sonar aided navigation for ship hull inspection, which extends our previous work on using imaging-sonar for harbor surveillance [46]. Here we estimate the full 6DoF pose of the vehicle and no longer assume a horizontal ground plane for the image registrations. In addition we describe an architecture that was developed to allow joint estimation using constraints from a sonar and a camera. For details on the vision system the reader is referred to [54, 55]. Second, we present a sonar based multi-session feature based navigation system for AUVs. In this work a REMUS 100 vehicle equipped with a forward-looking imaging sonar was used. The two vehicles used in these projects are shown in Figure 3-1.



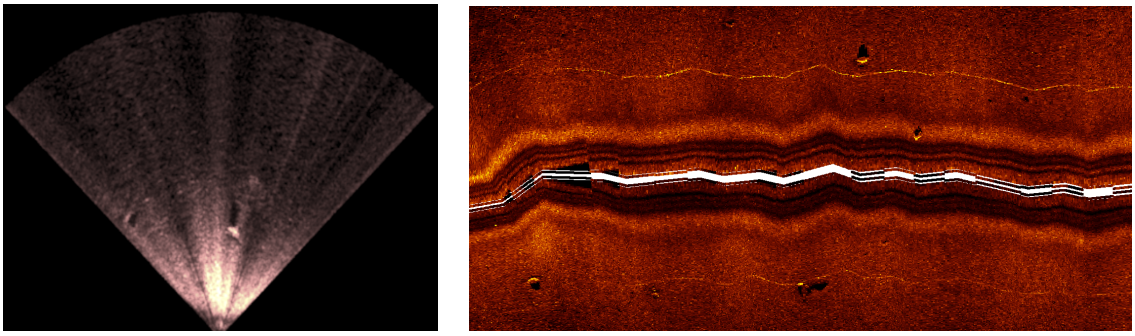
(a) Camera target (b) Sonar target

Figure 3-2: A comparison of a target seen by a camera and an imaging sonar.

3.1 Sonar Imaging

Light attenuates rapidly underwater resulting in limited range when using vision underwater. Particles in the water cause backscatter, making it difficult to see in turbid waters. On the other hand, sound propagates well underwater — even in turbid conditions — making the use of sonar systems common underwater. The drawback of using sonar includes lower resolution, lack of color information and often noisy images. A comparison of a camera and a sonar view of a target on a ship hull is shown in Figure 3-2. The camera gives much better details, while the target can be detected farther away using sonar.

A sonar works by transmitting a sound wave and then listening for the incoming sound as it is reflected from surfaces it hits along the way. The intensity of the return will depend on many factors: surface properties, incidence angle, distance, the transmission power, and the transducer beam pattern. For example, a muddy bottom will give a weak return, while the edge of a protruding rock will have a strong return and might also cast a shadow — depending on its height. These strong returns and shadows are common features that both human operators and detection



(a) Forward-looking imaging sonar (b) Side scan sonar

Figure 3-3: Example of different type of sonars.

algorithms use to analyze the sonar images. An illustration of the imaging process is given in Figure 3-4.

One of the earlier sonar systems used for imaging is the side scan sonar (see Figure 3-3(b)), which images a single line with each ping. On the other hand the sonars used in the work presented here sample from multiple directions simultaneously, e.g. the DIDSON sonar used in the ship hull inspection has 96 beams with 28 degree field of view and the BlueView has 512 beams with 90 degree field of view. These type of sonars are often referred to as imaging sonars, or multi-beam sonars when used for bathymetry measurements.

The beams of the imaging sonar provide a fixed field of view in front of it. The convention is a right handed coordinate system with x-axis pointing forward, y-axis to the right and z-axis down which is illustrated in Figure 3-5. The angle θ is a rotation around the z-axis giving the angle from the x-axis to a particular beam, and the elevation is ϕ which is the angle from the $x - y$ plane.

The measurements from the sonar are provided as intensity, a beam number, and range (strictly speaking, the range is provided as a time). Thus it can be convenient

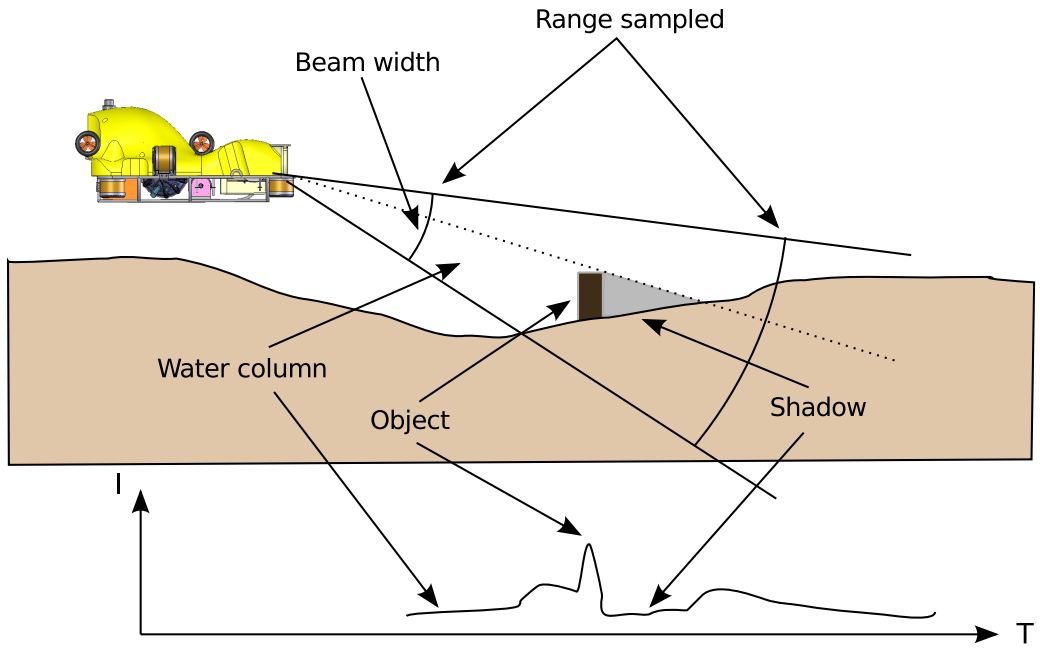


Figure 3-4: A high-level illustration of the sonar imaging process. First the sonar transmits an acoustic wave; the intensity of the reflected sound is then recorded at fixed intervals. The returned intensity will depend on various factors like surface properties, incidence angle, distance, the transmission power, and the transducer beam pattern. The plot on the bottom illustrates what the intensity might look like as a function of time along a single beam.

to think of the measurements in spherical coordinates $s = [r\ \theta\ \phi]^T$

$$\mathbf{s} = \begin{bmatrix} r \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2 + z^2} \\ \arctan 2(y, x) \\ \arctan 2\left(z, \sqrt{x^2 + y^2}\right) \end{bmatrix} \tag{3.1}$$

where $\mathbf{p} = [x\ y\ z]^T$ are the Cartesian coordinates of a point in the sonars frame of reference. The transformation from spherical coordinates to Cartesian coordinates is provided by

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r\cos\phi\cos\theta \\ r\cos\phi\sin\theta \\ r\sin\phi \end{bmatrix} \tag{3.2}$$

$$\tag{3.3}$$

One difficulty is that the sonar does not provide a measurement of the elevation ϕ. We address this problem in two different ways: a) in the ship hull inspection we assume a locally flat plane, b) in the feature based navigation we estimate the 3D feature position by minimizing the reprojection error for multiple measurements. In the next two sections we will look at these approaches in more detail.

3.2 Ship Hull Inspection

In this section we present an imaging-sonar aided navigation system for underwater ship hull inspection, which extends the work on harbor surveillance presented in [46]. The work presented here has been published in [42]. The contribution of this thesis to that work is the sonar processing, and an architecture for distributed state estimation

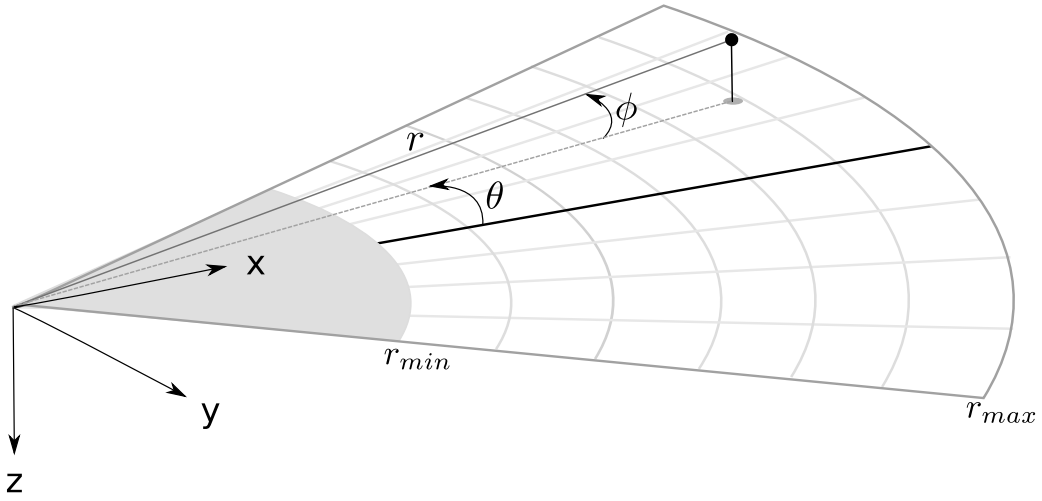
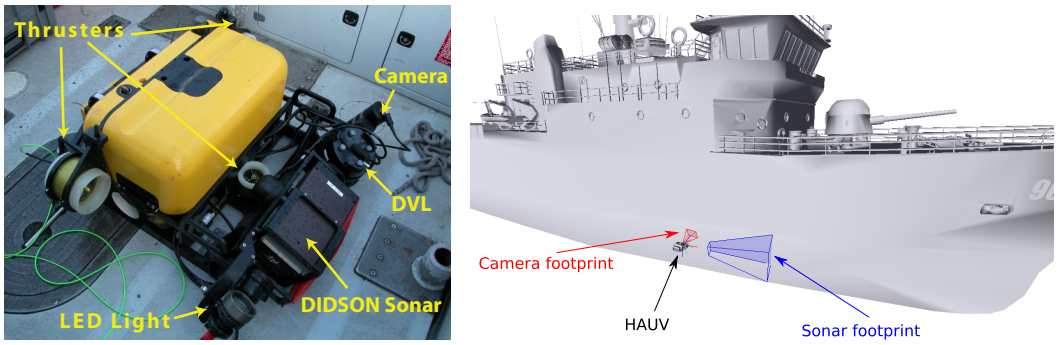


Figure 3-5: Imaging sonar geometry.

enabling the incorporation of camera constraints along with the sonar measurements.

The vehicle used for the ship hull inspection is the Hovering Autonomous Underwater Vehicle (HAUV). Its current generation is the Hull Unmanned Underwater Localization System (HULS) — developed by Bluefin Robotics — and is described in detail in [102]. It is equipped with a DVL, IMU, pressure sensor, DIDSON imaging sonar, and a camera. As shown in Figure 3-6(a) it has six thrusters, allowing the vehicle to hover in place; this is essential to accurately track the hull of the ship. When inspecting the non-complex part of the ship the vehicle operates in hull-relative mode and the actuated DVL is pointed toward the hull to keep a fixed standoff. The sonar points to the side of the vehicle, viewing along the hull. The camera is oriented in the same way as the DVL — pointing directly at the hull. The HAUV in relation to a ship hull and the sensor footprints are shown in Figure 3-6(b).



(a) The Bluefin-MIT Hovering Autonomous Underwater Vehicle (HAUV)

(b) The HAUV inspecting a ship hull

Figure 3-6: Ship hull inspection with the HAUV

3.2.1 State Estimation

The estimated vehicle state consists of its position and orientation at different times during the mission. The position is represented in 3D Cartesian coordinates (x, y, z) and the orientation as the Euler angles (ϕ, θ, ψ) for roll, pitch and yaw respectively. Internally during the optimization the orientation is represented as a unit quaternion. The vehicle coordinate frame is oriented such that x points forward, y to starboard and z points downward.

The pose graph can then be represented as a factor graph $G = (X, U, S, f, g)$ where $X = x_i | i = 1...N$ are the vehicle poses, $U = u_i | i = 1...N-1$ are the vehicle dead reckoning measurements, and S are the sonar measurements derived from aligning two sonar views. The factor f predicts the sequential motion

$$x_{i+1} = f(x_i, u_i) \tag{3.4}$$

and g the sonar measurement

$$s_k = g(x_{i_k}, x_{j_k}, \alpha, \beta) \tag{3.5}$$

where $s_k = [x, y, \theta]$ is the 2D transformation between the two sonar frames projected in the plane of the sonar, α and β are the two angles of the rotary actuators that are used to rotate the sonar.

This model is similar to the standard pose graph formulation. The variables of interest are poses along the vehicle trajectory, and the measurements are constraints between the poses. In this case the constraints from the sonar provide partial constraints, i.e. they are 3DoF while the vehicle pose is 6DoF. The dead reckoning from the vehicle provides 6 constraints between poses so the problem is well posed. In addition to the relative measurements the vehicle also has an IMU and a pressure sensor, allowing us to further constrain the depth, roll and pitch of each pose. Finally, the estimated state is provided by the maximum likelihood estimate of the distribution $P(X|U, S)$ and is computed using iSAM — a nonlinear least squares incremental solver.

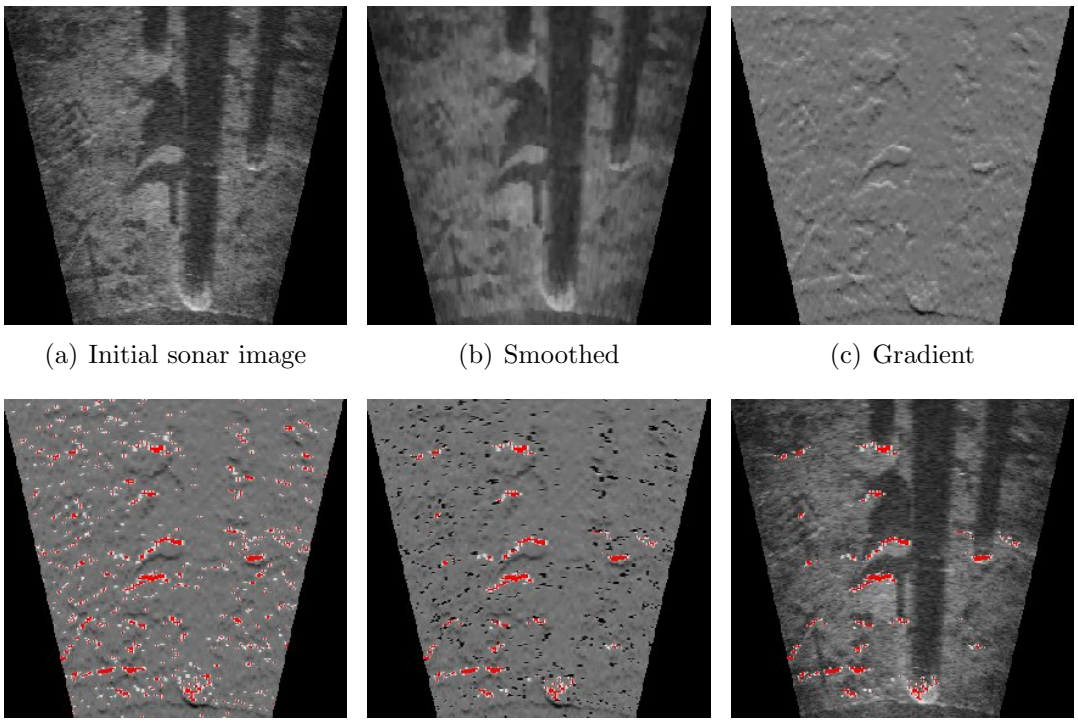
3.2.2 Sonar Frame Alignment

To correct drift in the navigation the current sonar image is aligned with a previously seen image. The approach taken is motivated by laser scan matching, which has been successfully used for ground vehicles. A set of dense features is extracted from each image and they are aligned using the normal distribution transform (NDT) scan matching algorithm [3].

Feature Extraction

The individual point measurements in a sonar image can be roughly categorized into background, highlights and shadows. The highlight normally comes from objects protruding from the surface which can cast a shadow on the background. The approach we take to feature extraction is to examine the intensity gradient along each sonar beam, allowing us to detect the edges of objects on the surface we are imaging. We look only at the gradients along the beams and detect transitions close to the object. This is to avoid effects from shadows from tall objects. A shadow farther out can move significantly with different viewpoints.

An example of the different steps of the algorithm is shown in Figure 3-7. For illustration purposes the images have been projected into 2D Cartesian coordinates. The input sonar image is shown in Figure 3-7(a). Initially the image is smoothed to remove high frequency noise (Figure 3-7(b)). Next, a gradient is computed along each sonar beam (Figure 3-7(c)). An adaptive threshold on the gradient is then used to select a fixed fraction of the points. This is to account for variability in the viewing angle and the objects inspected. If there are no objects in the image the top fraction tends to be spread around the image and is discarded during the clustering stage (Figure 3-7(e)). After the points have been selected based on the gradients they are projected into a 2D Cartesian image. The points are clustered, and points that have few neighbors are eliminated. The final result is shown in Figure 3-7(f). The output is provided as a collection of 2D points that are used to align with existing frames, or stored for later reference.



(a) Initial sonar image (b) Smoothed (c) Gradient

(d) Threshold (e) Clustering (f) Extracted Features

Figure 3-7: Intermediate steps of the feature extraction process. The extracted features are shown in red.

Alignment

After the features have been extracted they are projected into a 2D Cartesian coordinate frame. The Cartesian projection of the point $\mathbf{p}$ is provided by

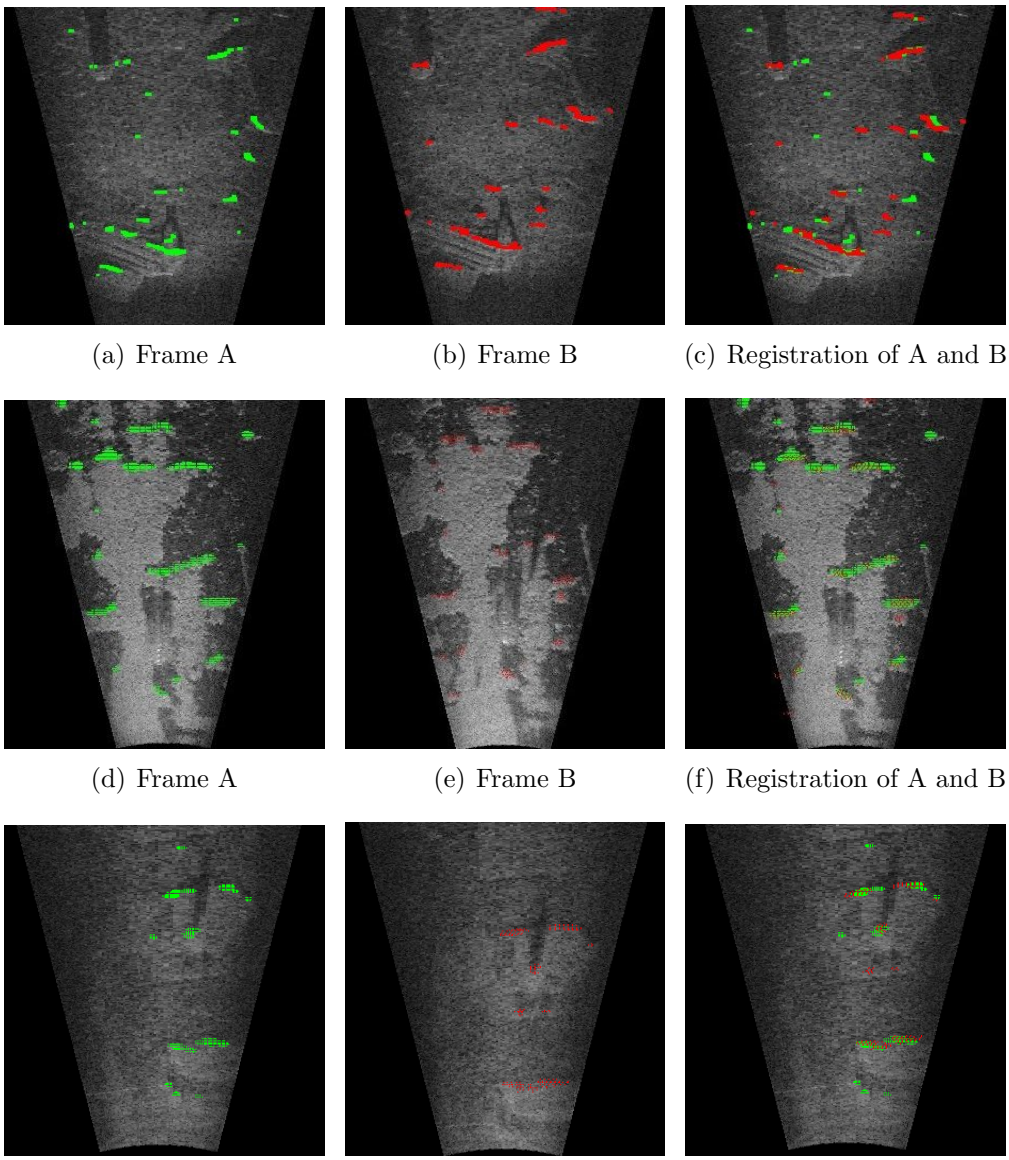
$$\hat{I}(\mathbf{p}) = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} r\cos\theta \\ r\sin\theta \end{bmatrix} \tag{3.6}$$

This projection can be viewed as an approximation to an orthographic projection looking down the z-axis. These points are then used for the alignment. As mentioned earlier the alignment algorithm used is the NDT algorithm. It works by constructing a grid over the scan and then, for each grid cell, the mean and covariance of the points in that cell are computed. This is called the NDT of the scan and serves as the model for the registration. The alignment is then computed by minimizing the cost between the incoming scan and the model. The cost of each point is the Mahalanobis distance (using the mean and covariance computed previously) from the cell the point is associated to. Examples of the alignment procedure, in different environments, are given in Figure 3-8.

3.2.3 Distributed State Estimation

In this section we describe the architecture of a distributed state estimation system that was developed for the ship hull inspection project. The motivation for this work was to enable the team from MIT and University of Michigan to work independently on the sonar and camera systems respectively while still sharing the same state estimation machinery and vehicle communication.

The system is structured into individual nodes: state estimation, sonar processing and camera processing. The nodes communicate using the lightweight commu-



(a) Frame A (b) Frame B (c) Registration of A and B

(d) Frame A (e) Frame B (f) Registration of A and B

(g) Frame A (h) Frame B (i) Registration of A and B

Figure 3-8: Registration of two sonar frames showing the extracted feature and the two scans before and after registration. Red points show the model scan, green features the current scan.The right-most column shows the two scans after registration. Each row is from a different environment. Top: The riverbed in the Charles River; there are many distinctive features. Middle: From the USCGC Venturous; there are a lot of barnacles on the hull making it very feature rich. Bottom: SS Curtiss; here the ship hull is very clean and the extracted features are primarily from structures under the hull.

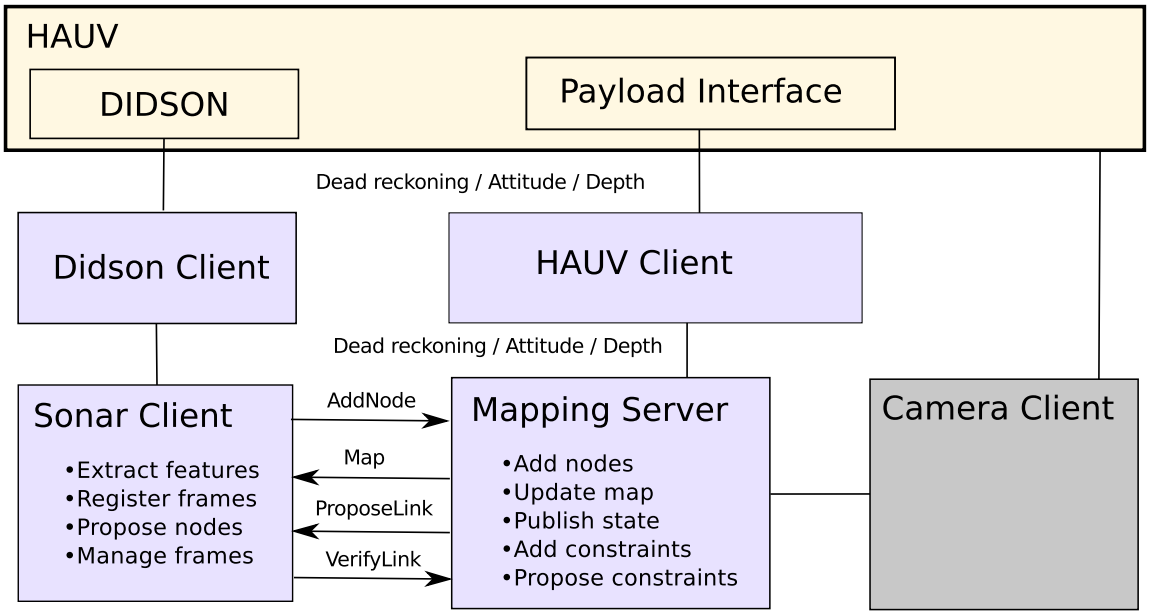


Figure 3-9: Overview of the state estimation architecture used for the ship hull inspection. The camera client was implemented by the University of Michigan and is not described here.

nication and marshaling (LCM) library [43] and can be distributed across multiple computers. In addition to nodes responsible for the state estimation there are nodes for communicating to the vehicle and the DIDSON sonar. An architecture diagram is shown in Figure 3-9.

The mapping server manages the map estimation. It receives vehicle navigation information from the vehicle and uses it to track the vehicle trajectory. Periodically the mapping server issues loop proposals if it is determined that the vehicle is within range of an existing view. This is determined by the distance between the current pose and another pose that is already in the map. When a sensor node receives a proposal, it uses the stored view information to verify the link, and report the transformation between the two views. In order to create a loop closure a node must be added to the pose graph. The sensor node can issue requests to the mapping server to add nodes at specific times. The sensor nodes are responsible for managing the sensor information required for loop closures. Additionally, the sensor nodes can also propose loop closures which enable sensor specific loop proposal mechanisms, similar to appearance based methods using vision.

This architecture worked well in building a shared state estimation. The LCM tools were particularly convenient for data logging and playback during development and testing of the system. In addition to the state estimation, a set of common visualization tools were developed using the *Libbot* library[1]. These tools were later extended to work as an operator console during the experiments, as described in the next section.

[1]Libbot is a library that was developed by MIT students during the DARPA Grand Challenge 2007. This code is available at `http://sourceforge.net/p/pods/home/Home/`. One of the components is a viewer framework that integrates well with LCM.

(a) *SS Curtiss.*

(b) *USCGC Seneca.*

	SS Curtiss	**USCGC Seneca**
Length	183 m	82 m
Beam	27 m	12 m
Draft	9.1 m	4.4 m
Displacement	24,182 t	1,800 t
Propulsion	Single-Screw	Twin-Screw

(c) Vessel characteristics.

Figure 3-10: Two vessels on which the HAUV has been deployed.

3.2.4 Results

To evaluate the system we used data from two experiments carried out on the SS Curtiss and the USCGS Seneca. In the experiments on the USCGS Seneca the system was run on-line in a combined mode, using both the camera and the sonar to aid the navigation. The benefit of the system was demonstrated by having a human operator set waypoints when interesting objects showed up on the sensor. Later the operator directed the vehicle to revisit some of these targets. The system successfully navigated to the object, illustrating the benefits of the online SLAM system. Finally, we evaluate the accuracy of the imaging-sonar aided navigation using a 2 hour sequence collected in the Charles River.

Ship Hull Inspection

During the course of the project, experiments were carried out on several ships. Here we will report on results from two of these experiments: the USCGS Seneca and the SS Curtiss. Information about these two vessels is provided in Figure 3-10.

During the operations on the USCGS Seneca the system was run on-line combining corrections from the sonar (MIT) and camera (U.Mich). The experiment was set up such that the vehicle started on a regular survey. A human operator monitored the information from the sensor. If interesting objects were observed she could mark the current position of the vehicle as a viewpoint. Later the operator could choose to direct the vehicle back to that waypoint by issuing a simple command. An illustration of the operator view is provided in Figure 3-11. The vehicle navigated accurately back to the target, demonstrating the benefit of having the online SLAM navigation.

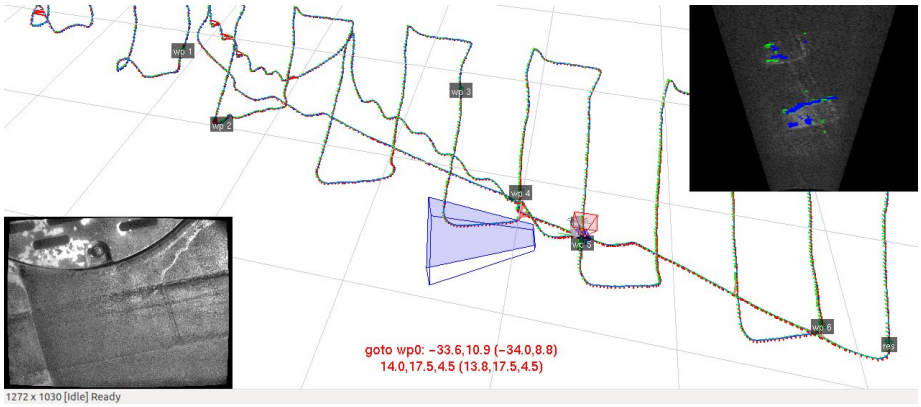
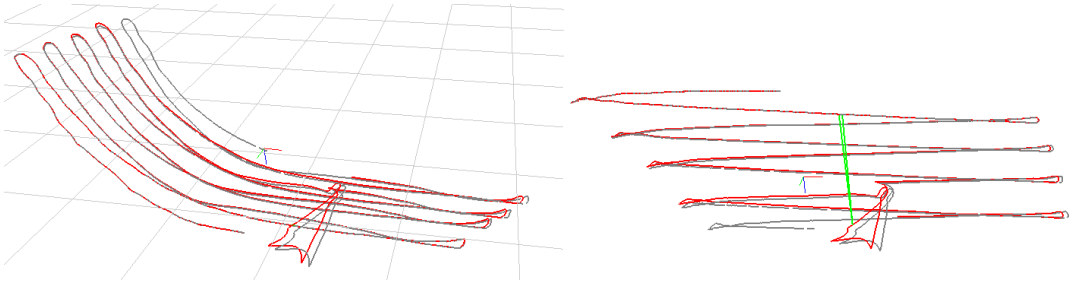


Figure 3-11: Operator view during a survey mission.

Examples of an estimated trajectory from the trials is shown for SS Curtis in Figure 3-12 and USCGS Seneca in Figure 3-13. The red lines are the smoothed trajectory. Gray lines are dead reckoning from the vehicle. Loop closures from



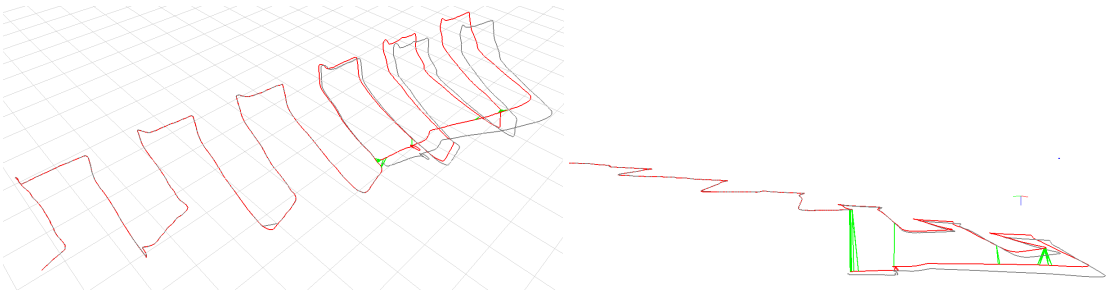
(a) Overview (b) Time elevation

Figure 3-12: A view of a pose graph from a ship hull inspection on the SS Curtiss. The red lines are sequential constraints between poses and the green lines are sonar based loop closures. The vertical axis is time, which better illustrates the loop closures.

the sonar system are shown in green. As the vehicle moves back it can align to a previous sonar frame. One of the major challenges when operating on clean ship hulls was that there are few features. (Contrast this to the feature rich hull of the USCGC Venturous shown in Figure 3-8(d)). Another problem was that many of the structures were highly repetitive making it necessary to keep tight bounds on the search region when matching frames. Addressing these issue are items for future research. Improving the feature extraction making it more discriminative is one direction. Also, matching over a larger region at a time as is done in the multi session re-localization presented in Section 3.3 might address to problem with repetitive structures. Finally, our collaborators on the HAUV project have presented promising results using camera and proposed methods to estimate visual saliency in camera frames [55].

Accuracy Evaluation

The accuracy of the imaging-sonar navigation was evaluated using a dataset collected in the Charles River (see Figure 3-17). The vehicle was configured with the DVL



(a) Overview (b) Time elevation

Figure 3-13: A view of a pose graph from a ship hull inspection on USCGS Seneca. The red lines are sequential constraints between poses and the green lines are sonar based loop closures. The vertical axis is time, which better illustrates the loop closures.

facing downwards and the vehicle moved horizontally at a fixed altitude. The estimated state includes only horizontal position and heading. This is a simpler setup than in the ship hull inspection case.

The dataset consisted of a 2 hour mission where the vehicle repeatedly surveyed a 20×15 meter rectangular area. The vehicle was looking under the pier at the MIT Sailing Pavilion and most of the loop closures occurred where the vehicle was close to the pier. An overview of the smoothed trajectory compared to the dead reckoning is given in Figure 3-14.

To evaluate the accuracy of the map estimate a selection of features in multiple frames were manually marked. These features are shown as markers in the overview image. To compare the estimated map to the DR position the features in each frame were projected using each position estimate, then compared to the position of the feature when it was first seen. As demonstrated in Figure 3-15 when using the smoothed estimate the re-projection error of each feature is bounded over the 2 hour mission time, while using the uncorrected navigation the error increases.

The HAUV must use a gyro for heading because it is designed to operate close to

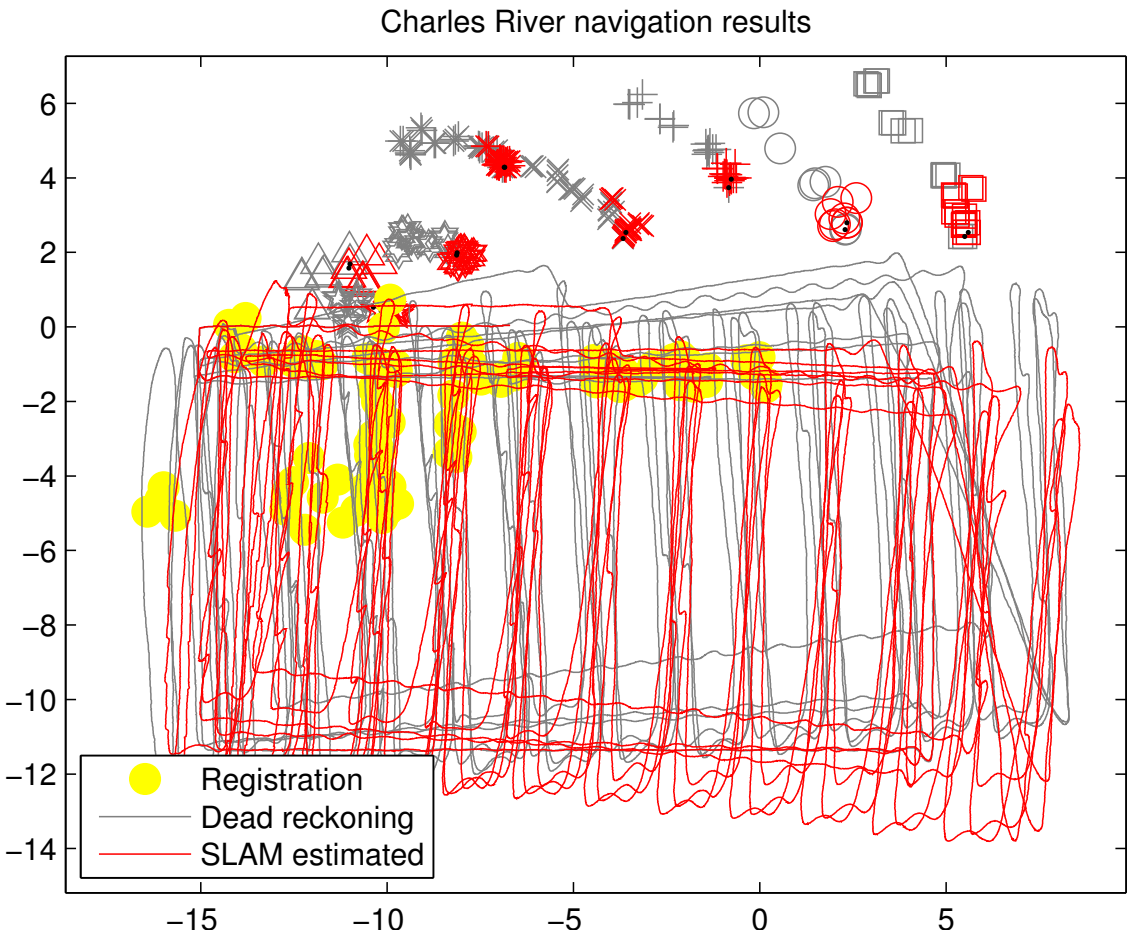
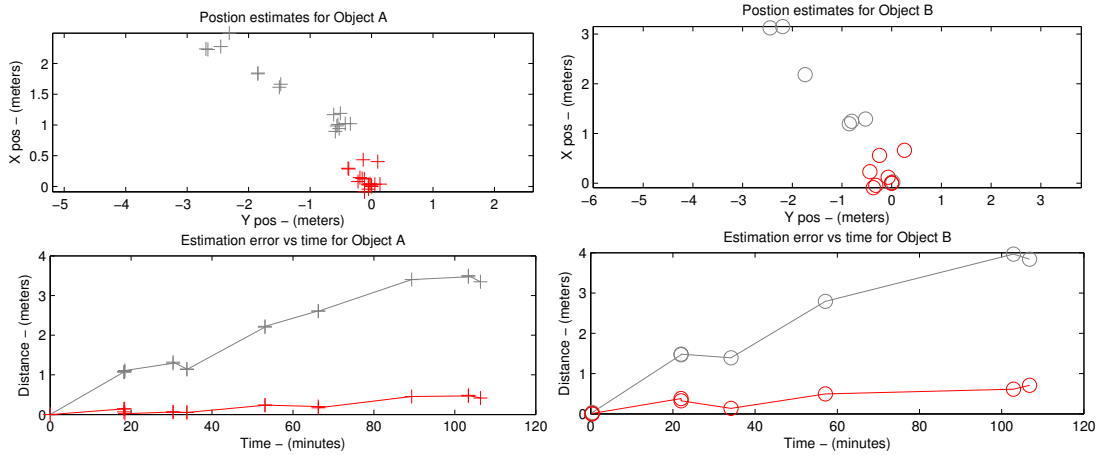
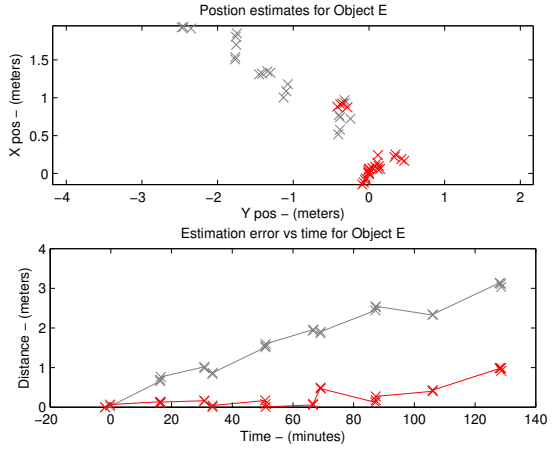


Figure 3-14: Overview of an inspection in the Charles river.

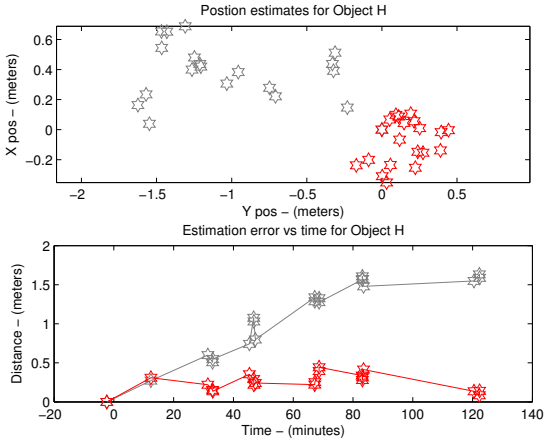


(a) Object 1

(b) Object 2

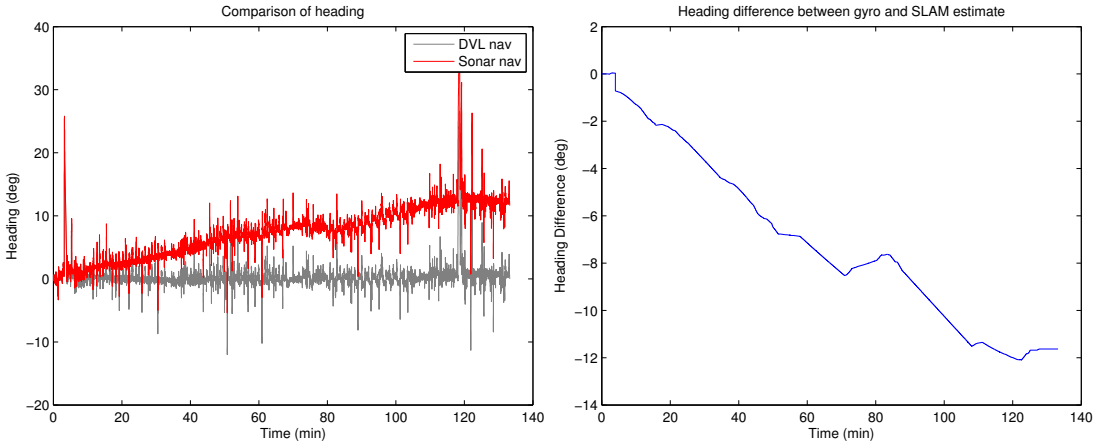


(c) Object 3



(d) Object 4

Figure 3-15: Showing the projected target location for a 2 hour mission



(a) Heading

(b) Heading error

Figure 3-16: Showing the heading drift over a 2 hour survey.

a ship hull which will interfere with a magnetic compass. This is a FOG so it is fairly accurate though not accurate enough to provide absolute heading. Thus over time the heading drifts as is shown in Figure 3-16. The heading drift is a large contributor to the navigation error for the HAUV vehicle.

We have demonstrated the accuracy of the system operating in the Charles River inspecting the river bed and pier structures. Further evaluation in the ship hull environment, beyond the on-line experimental results mentioned earlier, is a subject for future work. On the clean ship hull there is room for improvement in the feature extraction and handling large repetitive structures on the ship hull.

(a) MIT Sailing Pavilion

(b) HAUV sitting idle and preparing to enter the murky river

(c) Charles River mission control. Brendan and Hordur monitoring the mission progress

(d) HAUV breaching the surface

Figure 3-17: Operations at the MIT Sailing Pavilion in the Charles River

3.3 Multi-session Feature Based Navigation

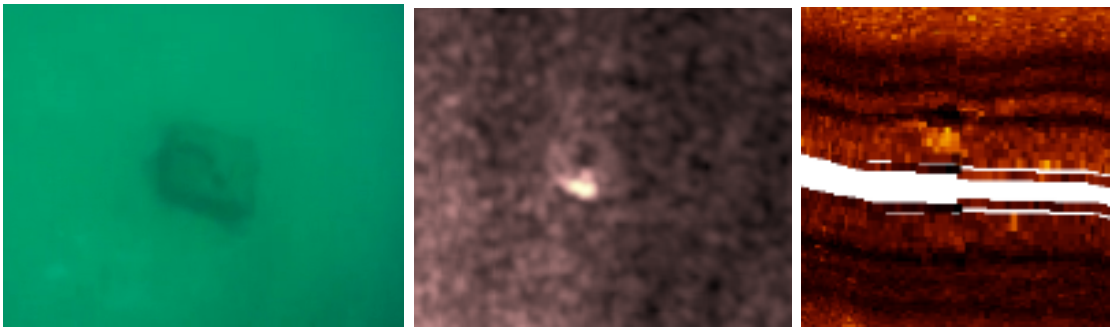
In this section we will describe a system for multi-session feature based navigation using a forward looking sonar. In seafloor surveying it is important to know the vehicle location. In particular, knowing the position relative to previous missions allows comparison of data between different surveys. It also allows for planing precise missions to inspect particular features in the area of interest, this could include portions of an oil-pipeline, target identification for mine countermeasures, etc.

In our work we build on methods previously developed in a project for underwater mine neutralization [29–31]. The goal of that project was to have a low end vehicle to use a forward looking sonar to home onto a target, attach to it, and finally neutralize it. An overview of experiments and results from the project is provided in [24]. In our work we are primarily interested in enabling navigation across multiple missions by global re-localization, continuous extension of the map, and finally a reduction of the map.

Next we will give the formulation used for the map estimation, then describe the main components of the system: target tracking, global alignment, and map merging. Finally we will provide results using data that was collected during experiments in May 2011, which took place in the Gulf of Mexico near Panama City. The vehicle used was a REMUS 100 vehicle equipped with a DVL and an RLG INS system. Additionally, the system had a Blueview Proviewer 900kHz imaging sonar, a sidescan sonar and a camera. Examples of data from these sensors is given in Figure 3-18.

3.3.1 Formulation

As in Section 3.2.1 the vehicle state is represented by the vector $x_i = [x\ y\ z\ \phi\ \theta\ \psi]^T$ where (x, y, z) are the position and (ϕ, θ, ψ) the orientation. We use the same right



(a) Camera (b) Imaging Sonar (c) Sidescan Sonar

Figure 3-18: A comparison of a target seen from different sensors.

handed coordinate system with x-axis forward and y-axis to the right as before. The map is represented as a factor graph $G = (X, L, U, Z, f, h)$ where $X = \{x_i\}$ are the poses, $L = \{l_j\}$ a set of points features, $U = \{u_i\}$ are controls, $Z = \{z_{i,j}\}$ are the feature measurements. The features are represented in global 3D Cartesian coordinates i.e. $l_j = [x\ y\ z]^T$.

The motion model is given by

$$x_i = f(x_{i-1}, u_{i-1}) + w_i \qquad w_i \sim \mathcal{N}(0, \Sigma_i) \tag{3.7}$$

with process noise w_i and the sensor model is

$$z_k = h(x_i, l_j) + v_k \qquad v_k \sim \mathcal{N}(0, \Lambda_k) \tag{3.8}$$

for feature l_j seen from pose x_i and v_k is the sensor noise. The difference here from the pose graph formulation mentioned earlier are the pose-to-feature measurements, rather than pose-to-pose measurements. To evaluate the measurement function h the feature l_j is first projected into the vehicle frame at time i using the pose x_i then into the sonar frame. Then using Equation 3.1 the spherical coordinates $[r\ \theta\ \phi]^T$

are computed. The sonar measures $[r\ \theta]$ but because the beam width of the sonar is limited we further constrain the elevation ϕ using a cubic normal parametrization as proposed in [29].

The estimate of the map is given by the ML solution for X and L given all the measurements Z, and as before computed by solving the associated nonlinear least squares problem.

For the multi-session operations the system maintains three estimates of the environment: 1) the prior map that was initially loaded, 2) the current estimate, 3) the merged map from the best global alignment. Each of the maps is a factor graph as described above. If the feature assignments from the global alignment system change, the merged map is discarded and a new map is formed with the new assignments.

3.3.2 Feature Tracking and Initialization

In this section we describe how the initial detections from the feature detector are tracked and initialized before they are passed on to the mapping system. The features are detected in the sonar image using the algorithm developed by Folkesson et al. [31] and are provided with the dataset that was used in this work. The detections find bright targets in the sonar image, and the measurement is given in range and bearing relative to the sonar. Because of noise in the sonar image the detector can return spurious measurements so we need to track and associate detections across multiple sonar frames before they can be passed to the mapping system.

The approach we take here is to first define a collection of feature tracks. Each feature track maintains a set of associated measurements, 3D positions, and uncertainty in the current vehicle frame. The track is then updated using an unscented

Kalman filter (UKF) [49].

First, each feature track is propagated into the current vehicle frame. Next, a predicted measurement $\hat{z} = [r\ \theta]^T$ and its associated covariance Λ is computed for each track. This is done using the unscented transform (UT) [49]. Then each measurement is associated to its nearest neighbor determined by the Mahalanobis distance $||z_k - \hat{z}||_\Lambda$ of the innovation. The match is only accepted if the distance is within a given threshold. Here we set the threshold to reject measurements outside the 99th percentile.

Each active matched track is then updated using the UKF update rules. If a track has received more then a certain number of measurements it is sent to the mapping system with all the measurements included to date. In subsequent steps new measurements for that track are passed directly to the mapping system. Tracks that fall behind the vehicle are removed from the tracker, in order to limit the number of active tracks at any given time. Next we will look at how the current map can be aligned and merged with the prior map.

3.3.3 Global Alignment

Periodically all possible assignments between the prior map and the current map are tested. The best possible map is then reported to the mapping system which can then use those feature assignments to merge the prior and current map estimates. The best possible match is chosen as the match that has the highest number of matched features.

To align the current mission to the prior mission we used the fact the vehicle navigation is accurate locally and provides accurate heading. In that case a single putative match is sufficient to compute a proposal for the alignment. Thus, instead of

using random sampling, as in RANSAC [27], to find a match, we can quite efficiently consider all correspondences.

To align two maps we consider the feature locations in each map $L = [l_0\, l_1\, \ldots\, l_N]$ and $L' = [l'_0\, l'_1\, \ldots\, l'_M]$. We want to find a transformation Δ that maximizes the number of matching features. We start by considering every pair (l_i, l_j) as an initial proposed alignment model. The initial proposal is then $\Delta_0 = l_i - l'_j$. A set of putative matches is found by transforming all the points in L' by Δ_0. A match is found by associating each point to its nearest neighbor. If the distance is within a given threshold it is accepted as a match.

If we assume that n landmarks are matched and they have been ordered such that l_k and l'_k have been matched together, where $k = 1 \ldots n$. Then we like to find a transformation Δ^* between the landmarks such that the distance between the two sets is minimized, and is given by

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum ||l_k - (l'k + \Delta)||^2 \tag{3.9}$$

$$= \frac{1}{n} \sum_{k=1}^{n} (l_k - l'_k) \tag{3.10}$$

Finally, the estimated alignment with the highest number of matched features and the lowest error is sent back to the mapping system. The mapping system then uses the transformation and the correspondences to merge the maps.

3.3.4 Map Merging and Reduction

After the alignment the correspondences between the prior map and the current map are used to create a combined map of the two. This combined map is eventually saved at the end of a mission to be used for subsequent runs. That way the map can be

gradually extended over multiple missions. We compared two different methods of combining the map. The first approach is to simply incorporate all the information from both maps, while the second tries to only incorporate information to add newly seen landmarks into the combined map.

In the full approach we start by creating a copy of the prior map. Then all the features from the current map that were not matched are added to the new map. In the new map their position is initialized using the computed alignment. Next all the poses from the current map are added to the combined map, and the poses are initialized using the estimated alignment. Finally all the measurements from the current map are copied over, and the measurements that measure matched features are updated accordingly.

The reduced approach starts in the same way by copying the prior map and then copying all the new features. Not eliminating any new features is one of the requirements for the reduction. The graph is reduced by incorporating information that is sufficient to include the new features and also constrain the trajectory to the existing map. First we examine all the measurements and mark the poses if they connect to a new feature or an existing feature. Then we go sequentially through all the poses. If a new feature is measured from a pose we require that n nearby poses measuring existing features are included in the graph. In our experiments we used $n = 20$ which proved sufficient to adequately constrain the newly added section. The parts that are removed are chains of poses that see only existing features.

This approach reduces the map by avoiding adding new information to parts of the map that have already been included. In Chapter 5 we will look at a different approach that reduces the graph yet still continuously integrates new information.

3.3.5 Results for Multi-session Feature Based Navigation

The feature based navigation system was developed and evaluated using data collected during experiments in Panama City, Florida in May 2011. The vehicle used was a REMUS 100 AUV equipped with a Blueview Proviewer 900kHz sonar. The purpose of the experiments was to demonstrate target acquisition given a prior map. The missions were repeated traversals through the same area allowing us to use the data to test our multi-session algorithm. An example of the survey area is illustrated in Figure 3-20. The top figure is a sidescan sonar image of a part of the area and the bottom figure shows the vehicle trajectory in that same area. Green points are tracked features. Red are features that have not yet been accepted. Gray points indicate detections not upgraded to a feature, clearly showing how noisy the initial detections are.

The vehicle also had a GoPro camera mounted under the vehicle to document captures. We calibrated the camera and synchronized it with the vehicle time. This allowed us to project tracked features onto the camera as shown in Figure 3-19. It is clear that the detections group around a target lying on the seabed.

Alignment to prior mission

First we look at the benefits of re-aligning the vehicle location to a map from a previous mission. Figure 3-21 shows an example of two missions. At the top is the initial mission. The grid size is 20m so the whole trajectory is around 160 meters across. The green line shows some of the feature measurements, the red triangles are features, and the vehicle trajectory is shown in red. Then a second mission is run (shown in the middle figure). The green points are features from the second missions, and green lines show the matches given by the alignment algorithm. The

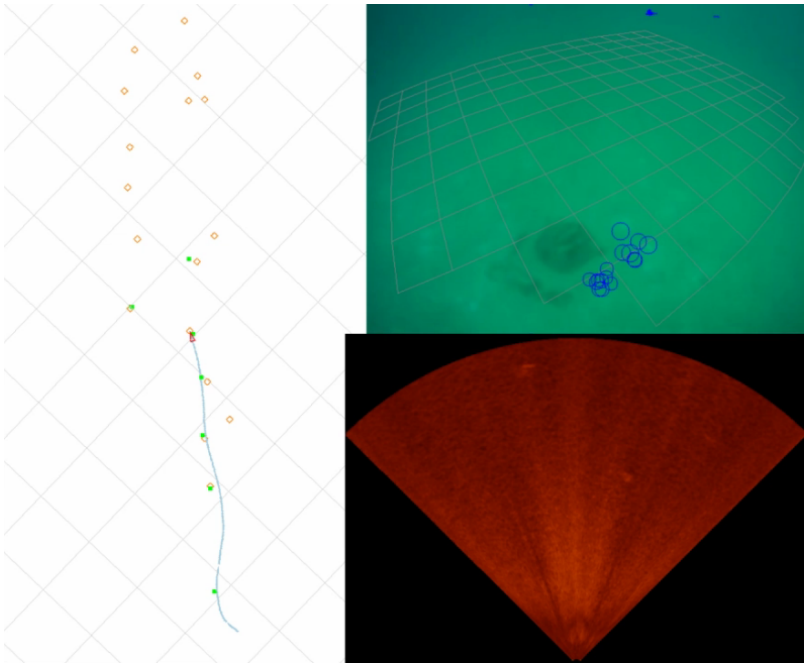


Figure 3-19: Fused view of sonar and camera.

error between these comes from both initial GPS error and then navigation drift as the vehicle approaches the field. Finally on the bottom the green trajectory is now the second mission aligned to the first. Now the features between the two missions line up.

Map merging and reduction

To compare the full and reduced map merge algorithms we selected 6 missions (shown in Figure 3-22) from two separate days. The green points are the feature estimates in the final combined map. The triangles are feature estimates from the individual missions. Each of the trajectories are shown in different colors and we see that each missions aligns well with the combined map. For the evaluation we considered how much the graph was reduced, the difference between the poses in each estimate, and

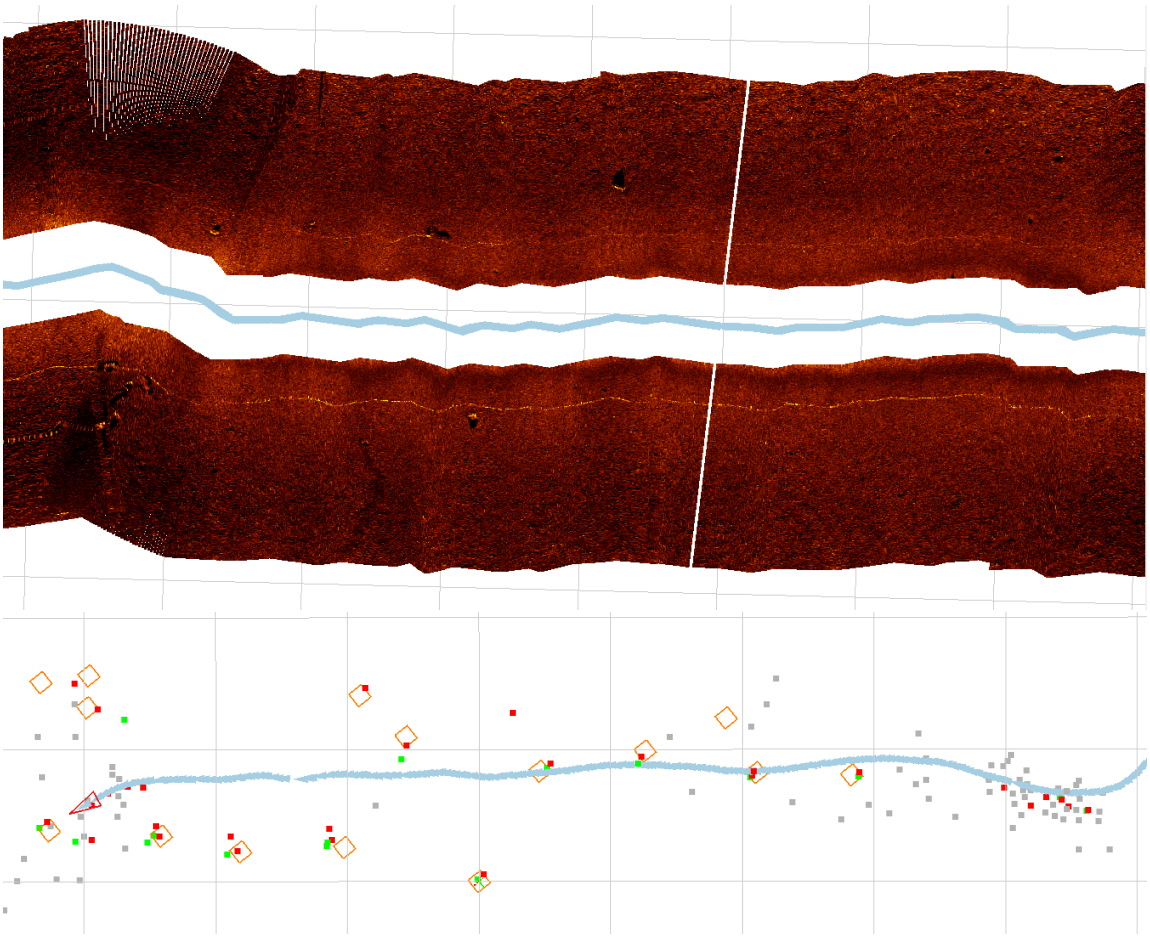
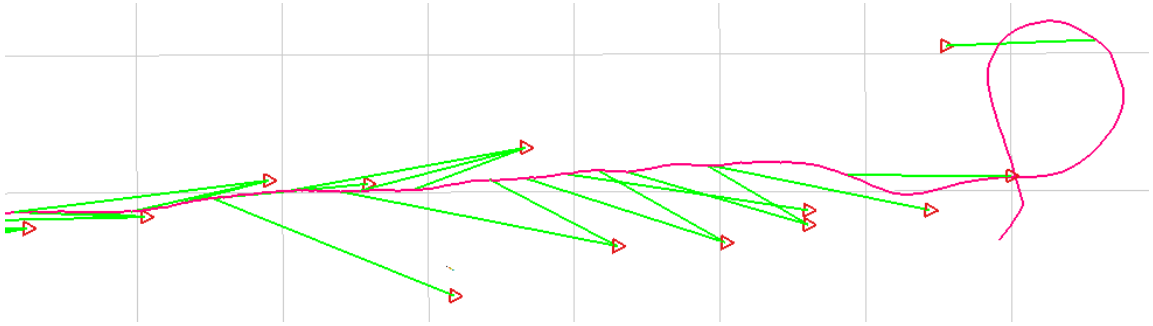
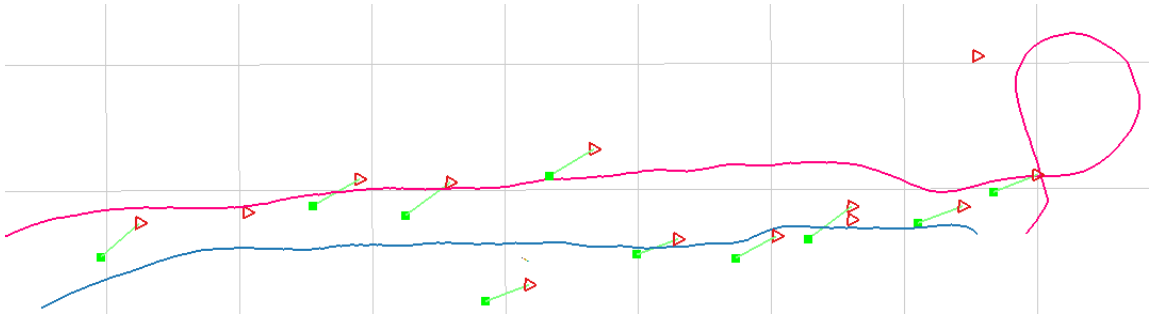


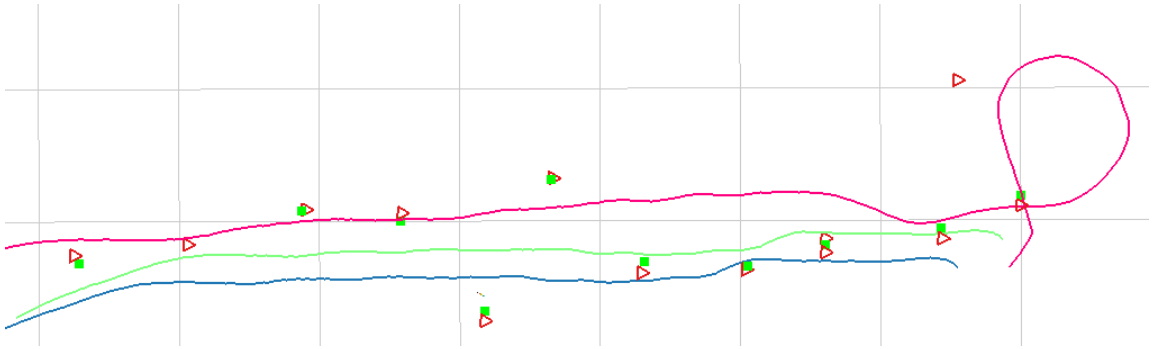
Figure 3-20: This figure shows the result of the estimated trajectory from the Panama City 2011 experiments. The boxes are objects identified from a previous mission. The green dots show the estimated position of features automatically detected in the forward looking sonar. The red points are features that did not get enough support.



(a) Initial mission



(b) Second mission



(c) Second mission after alignment

Figure 3-21: At the top is the initial mission. In the middle is a second mission where the missions have not been aligned. At the bottom is the result after alignment.

finally the timing information for individual components. The total mission time was around 19 minute or 2-4 minutes per mission.

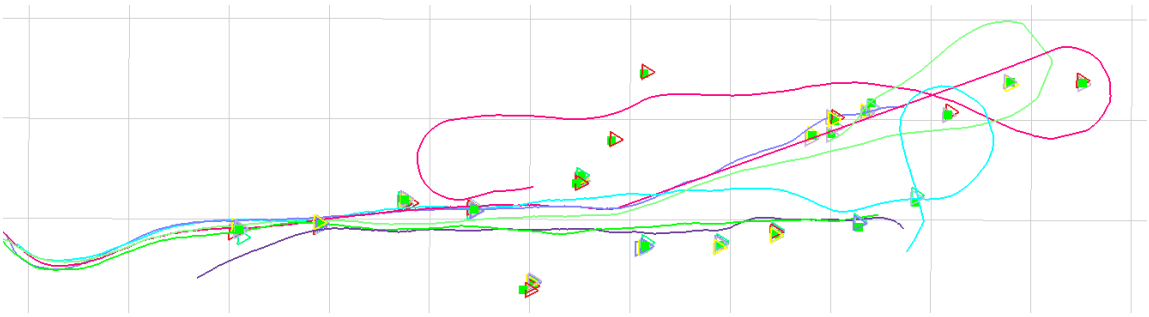


Figure 3-22: An overview of the 6 missions used for the evaluation.

The combined map constructed using the full merge includes in total 1520 poses and 30 features. For the reduced merge the final map consisted of 749 nodes and the same number of features. Figure 3-23 shows how the size of the map evolved over all the missions. The sharp edges in the graph is when the maps are first combined. The combined map is only constructed if the alignment procedure finds at least 4 matches. The size of the map was recorded each time the merge procedure was run. As expected the number of features quickly evens out because most of the mission time is spent in the same area. We also see that the growth of the reduced map follows the addition of new features while the full map continuously grows.

To evaluate the accuracy we looked at the average distance between poses in the two final combined maps. The mean distance was 0.36m with standard deviation 0.4m so some accuracy is lost with the reduction. The resulting trajectories are shown in Figure 3-24. Possibly this can be improved further by including more measurements between the missions, or requiring some minimal number of features to be matched between the different maps. In Chapter 5 we consider a different reduction approach that is applicable to pose graphs.

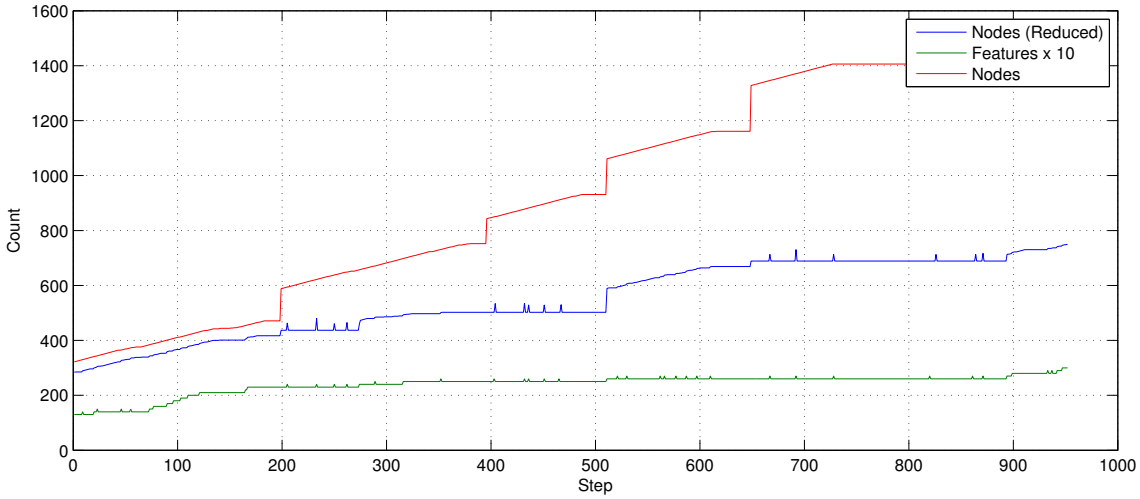


Figure 3-23: Number of features and poses in the reduced map.

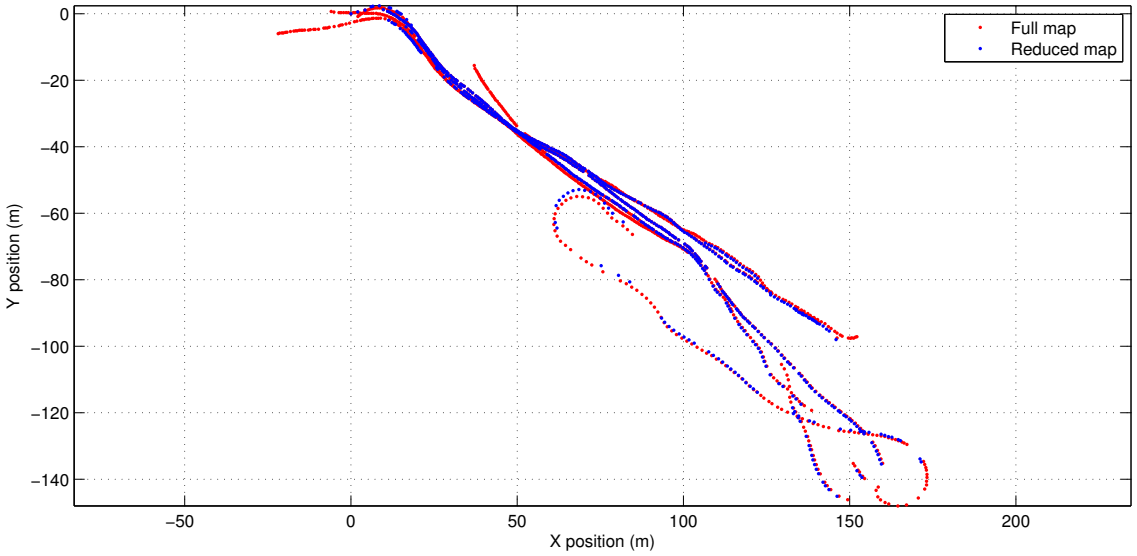
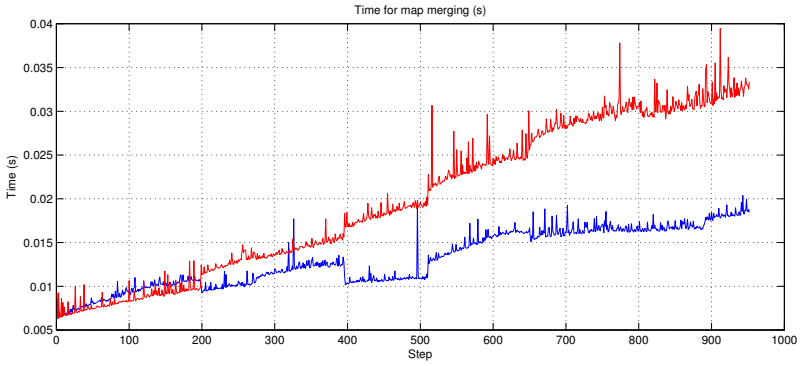


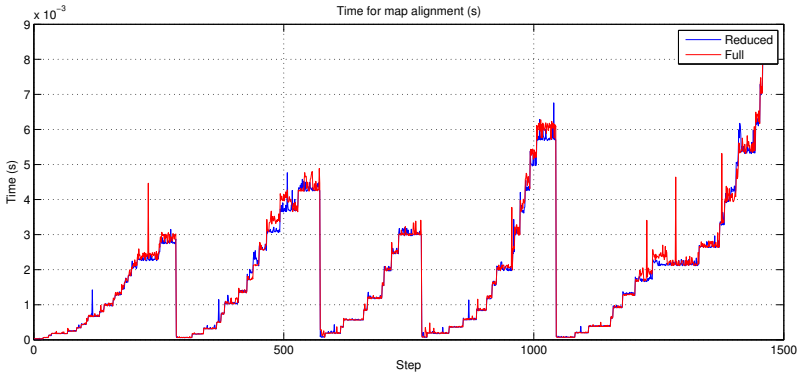
Figure 3-24: Comparison of the trajectories generated using the full and the reduced approach. The average distance between poses in the two trajectories is 0.36m with standard deviation of 0.41m.

The time for the map alignment, map merge, and map update was measured while constructing the maps. For the timing experiments a Intel(R) Core(TM) i7-2920XM CPU @ 2.50GHz machine was used. The computation time for the duration of the missions is shown in Figure 3-25. These numbers are single point measurements for a single execution. The map update takes most of the time. The optimization was run in batch mode because the alignment can change the feature correspondences and the resulting merged map. An interesting direction for future work is to consider how to incrementally update a combined map as the correspondences change, and when variables are removed or added.

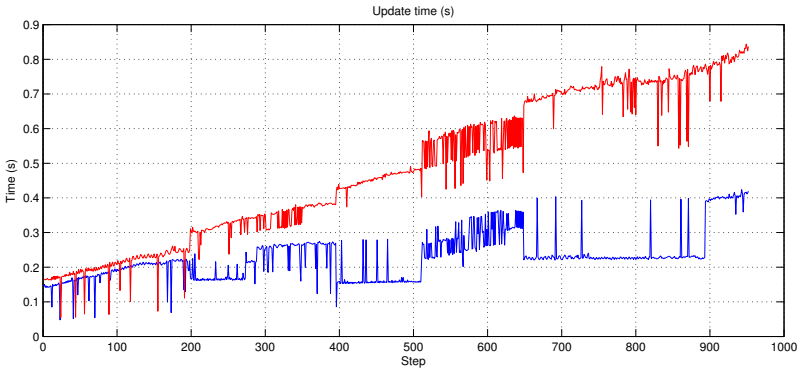
In the final stage the map merge takes around 20ms for the reduced map and 33ms for the full map. The map alignment is very similar for both maps, because the complexity is governed by the number of features and the number of poses. For the small map size considered here the alignment does not take much time. The alignment algorithm considers $O(MN)$ pairs, assuming the maps have M and N features, so as the maps gets larger the computational cost might become significant. One approach would be to consider only a fixed number of features in the current map, e.g. only match a local neighborhood around the current position in relation to the prior map.



(a) Map merging



(b) Map alignment



(c) Map update

Figure 3-25: Timing comparison of the full and reduced map estimates. At the top is the merge of two maps. In the middle is the alignment algorithm. The bottom plot shows the time for update step taken by the nonlinear solver.

Chapter 4

Real-time Visual SLAM

Cameras are convenient sensors to use for SLAM. They are small and inexpensive compared to other sensors, e.g. lasers. In addition they give rich information about the environment which can be used to identify specific places and recognize objects, in addition to estimating the sequential motion of the camera. In this chapter we will describe in detail a SLAM system that uses vision as its primary input and map representation. An architecture overview of the system is given in Figure 4-1. The main components are: visual odometry, loop proposal, loop closure, map management, and map estimation. These components will be described in detail in the following sections.

The map representation is a set of keyframes, their poses and features associated with them. The primary motion estimate is visual odometry which is computed by matching consecutive frames. A global loop proposal and frame alignment algorithms are used to correct errors accumulated from the odometry measurements. The SLAM system achieves real-time performance and can use either stereo or RGB-D cameras. In addition to using the camera for motion estimation an IMU and wheel odometry

are used. This improves the overall robustness of the system because there are many situations that will cause the vision system to fail. Finally, the vertical acceleration measured by the IMU is used to track elevator motion, enabling operations in multi story buildings.

4.1 Map Representation

The map is represented as a collection of camera poses and features from the camera frames associated to each pose. Combined, the poses and the features can be used to form a global map that can then be used for path planning and navigation. The location of each feature in the image and a descriptor is stored in the map. The descriptor is later used for both discovering and verifying loop closures, as described in Section 4.6. Also, to efficiently search for similar places an inverted index of a bag of visual words is maintained. In addition to the camera poses a collection of constraints on the poses are also kept with the map. Together the poses and constraints form a standard pose graph formulation [82] of the map.

The poses are 3D poses represented by $\mathbf{x} = [x\ y\ z\ \phi\ \theta\ \psi]$ which are the Euclidean position of the pose and the orientation using Euler angles representing roll, pitch and yaw. The primary constraints are 6DoF rigid body constraints between a pair of poses which can be derived from wheel odometry, visual odometry, or alignment derived from visual loop closures. When an IMU is available an absolute constraint on the vertical component of the pose is added, which is based on measurements from the acceleration sensors.

As described in Section 2.4.2, because the measurements are noisy we model them as probability distributions. The Maximum Likelihood (ML) estimate is then computed and used as the current estimate of the map. Let $\mathbf{X} = \{\mathbf{x_i} | \mathbf{x_i} \in \mathbb{R}^6, i =$

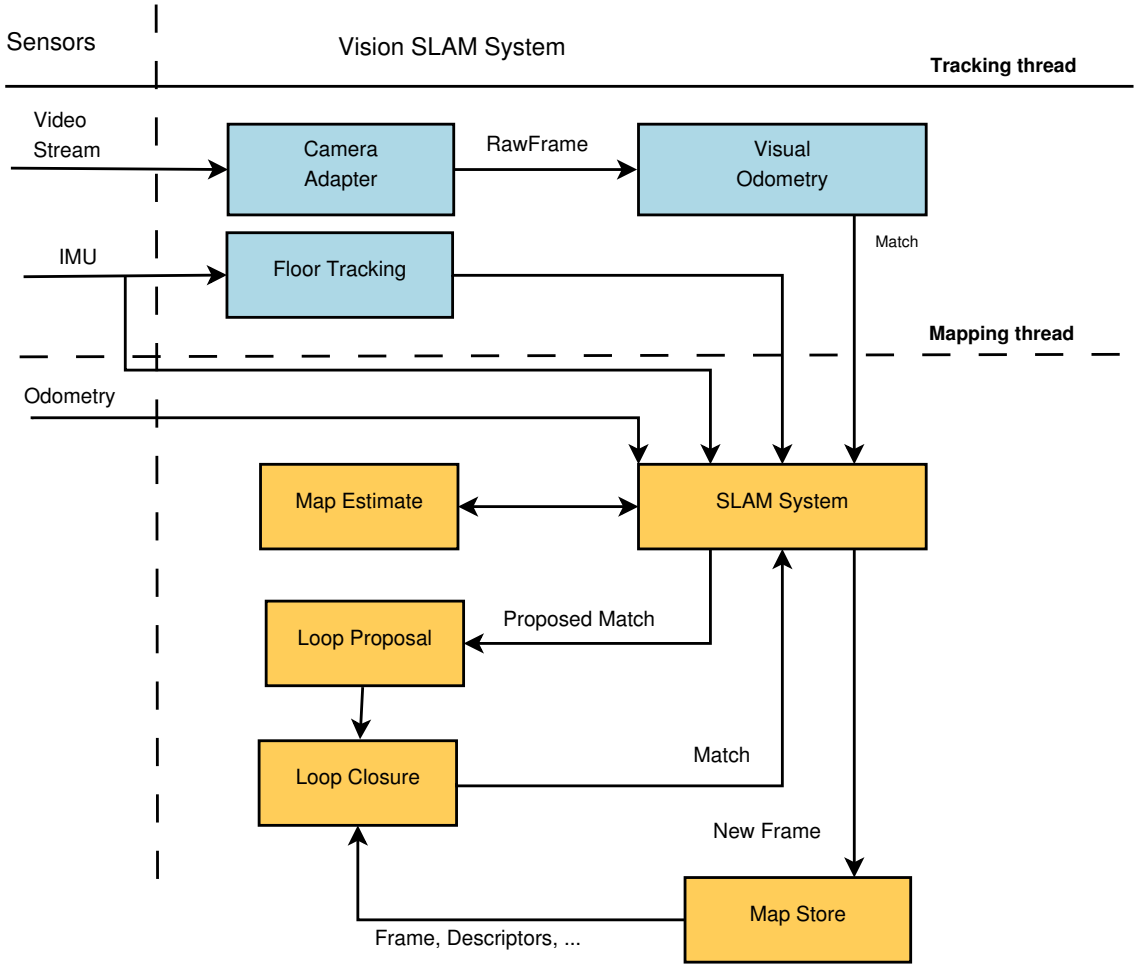


Figure 4-1: Architecture for the visual SLAM system. On the left are the sensor inputs and on the right are the clients that consume information from the visual SLAM system. The FOVIS library is used for visual odometry [44] and geometric verification of loop closures. Dynamic Bag of Words [84] has been used for loop proposals. The OpenCV library is used for computing feature descriptors and matching. The map estimation uses the iSAM library.

$1...N\}$ be a set of N poses and $\mathbf{F} = \{f_k(z_k, i_k, j_k, f_k)|k = 1...M\}$ be a set of M factors corresponding to constraints between poses. Then the ML estimate X^* is given by

$$X^* = \underset{X}{\operatorname{argmax}} F(X) = \underset{X}{\operatorname{argmax}} \prod_{k=1}^{M} f_k(X_k, x_k) \tag{4.1}$$

where X_k is the subset of variables connecting to factor f_k.

The measurements $\mathbf{z}_k$ are random variables s.t. $f_k(\mathbf{x}_{i_k}, \mathbf{x}_{j_k}) = \mathbf{z}_k + \mathbf{w}_k$ and $\mathbf{w}_k$ is the measurement noise. If we model $\mathbf{w}_k$ as being a zero mean Gaussian random variable then the maximum likelihood can be computed by solving a nonlinear least square problem. For robust cost functions a weighted least squares method can be used [40].

4.2 Alignment

When using only vision the constraints z_k will be derived by aligning two nearby frames. Lets assume we have a set of point measurements a_i and b_i that measure point p_i in frame A and B respectively. We like to estimate the transformation T between the two frames. The transformation T will take a point in frame B and transform it to frame A

$$\mathbf{p}_i^a = T\mathbf{p}_i^b \tag{4.2}$$

and $T = [R|\mathbf{t}]$ where R is a rotation matrix and the vector $\mathbf{t} = [xyz]^T$ is the translation. We parametrize the transformation with $\mathbf{x} = [xyz\phi\theta\psi]$ where x, y, z is the translation and ϕ, θ, ψ are Euler angles corresponding to roll, pitch and yaw respec-

tively.

To find the alignment $\mathbf{x}$ we find a transformation that minimizes the Euclidean distance between the points

$$\hat{\mathbf{x}} = \underset{X}{\operatorname{argmax}} \sum_i ||\mathbf{p}_i^a - T\mathbf{p}_i^b||^2 \tag{4.3}$$

and assuming additive, zero-mean Gaussian noise, then $\hat{\mathbf{x}}$ is the ML estimator. Because the 3D position is computed by triangulating the points from each stereo camera a better estimate is achieved by minimizing the reprojection error [79].

$$\hat{\mathbf{x}} = \underset{X}{\operatorname{argmax}} \sum_i ||\mathbf{z}_i^a - f(KT(\mathbf{x})\mathbf{p}_i^b)||^2 + ||\mathbf{z}_i^b - f(KT(\mathbf{x})^{-1}\mathbf{p}_i^a)||^2 \tag{4.4}$$

where K is the camera matrix

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

and $f : \mathbb{R}^3 \to \mathbb{R}^2$ projects a 3D point onto the image plane.

$$f(\mathbf{p}) = \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix} \tag{4.6}$$

This is the approach used to estimate both sequential motion and loop closure constraints in sections to follow.

4.3 Visual Odometry

The incremental motion is estimated by matching sequential video frames and computing the motion between them. This is referred to as visual odometry (VO) [79]. There have been many implementations of VO in recent years. We use a publicly available implementation called Fast Odometry for VISion (FOVIS) [44]. It supports both RGB-D and stereo cameras. In the reminder of the section we will give a description of this particular algorithm and how we combined it with the overall vision SLAM system.

The main steps for a VO algorithm are:

1. Detect keypoints

2. Extract features

3. Match features to previous frame

4. Compute motion between frames

Following is a description of how FOVIS implements these steps. The input to the motion estimation algorithm, when using a stereo camera, are the gray colored frames for the left and right camera; when using the RGB-D camera the input are a gray image and a depth image. The output from the algorithm is the motion estimate from the previous keyframe to the frame passed in. Possibly the keyframe was changed to the most recent frame passed in. In addition it is possible to retrieve a covariance estimate and the detected features.

Parameter	Descriptions	Typical
FAST threshold	Determines which keypoints are used	10
# inliers for keyframe change	A keyframe is changed if the number of inliers drops below this threshold	100
# inliers for motion estimate	If the number of inliers is below this threshold then the motion estimation is not computed and it is assumed the VO has failed	15
Minimum reprojection error	After the optimization all points that are within this threshold are determined as inliers	1.0
Grid size	Used to ensure an even distribution of features across the image. A fixed number of features with highest FAST score are selected from each grid cell	80x80
Clique inlier threshold	The difference of the distance between a pair of features so they are determined as a valid pair	0.15

Table 4.1: Parameters for the visual odometry module.

Parameters

There are many parameters that will affect the performance of the visual odometry. A description of the parameters and typical settings is provided in the table below.

Additionally there are specific features to choose from, e.g. the grid can be enabled or disabled, the FAST threshold can be fixed or adaptive, sub-pixel refinement can be enabled. The parameters given in Table 4.1 are those that worked well for our scenario. We used a fixed FAST threshold instead of using the adaptive threshold that is supported, because in some cases the threshold would get too low, allowing many spurious features to be selected, and decreasing the overall accuracy. Having a minimum setting on the adaptive threshold might minimize that problem.

Detect Keypoints

First a Gaussian pyramid is constructed from the input image. A FAST [90] detector is used to find keypoints on each level of the image pyramid. The FAST detector works by finding a segment of points on a circle around the pixel being tested that

have intensity that is lower or higher then a given threshold. If the length of this segment is greater then 9 pixels the pixel is classified as a keypoint. For increased stability, maximal suppression is also performed in the neighborhood of the detected keypoint. This algorithm is extremely fast but it is sensitive to noise in the image.

Feature Extraction and Matching

After the keypoints have been detected each one is matched to the neighboring keypoint that is most similar; the neighborhood is defined as all keypoints within a fixed radius in the image. The image patch around the keypoint is used as the descriptor and the similarity is determined by the sum of absolute differences (SAD). A match is accepted if the match from reference frame to target frame, and target frame to reference frame are mutually consistent. After the match, a sub-pixel refinement is performed on the target keypoint, by aligning the gray image patches. The gray image patches are aligned by minimizing the sum-of-square errors between the patches. An example of frame-to-frame feature tracking is given in Figure 4-2(a).

Compute Motion Between Frames

The cameras supported by FOVIS are either RGB-D or stereo cameras, and the only keypoints considered are keypoints that have depth. Having the depth information is particularly useful for both determining inliers and initializing the motion. To determine the inliers the distance between a pair of keypoints in the reference frame is compared to the distance of the corresponding keypoints in the target frame. The transformation being sought is a rigid body transformation which preserves distances. Thus if the distances are within a pre-determined threshold then these correspondences are marked consistent. This forms a graph over keypoint correspondences.

Now a maximal clique is determined in the graph. This clique forms the inliers set.

Given a collection of 3D points and consistent correspondences it is possible to use direct methods like Horn's absolute orientation algorithm to compute the initial motion — as is done in this work.

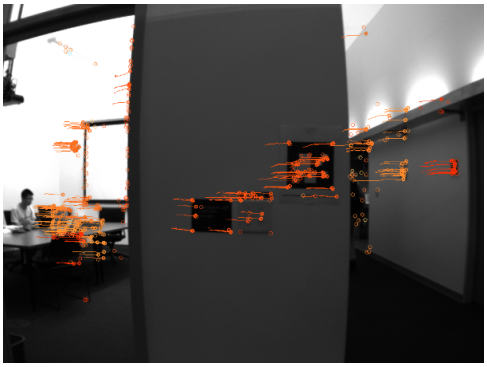
This initial estimate is further refined by minimizing the bi-directional reprojection error. This is done by solving a nonlinear least squares problem using LevenbergMarquardt (LM). After the optimization the re-projection error is computed for each point. If the re-projection error exceeds a given threshold the point is removed from the inlier set. After the outlier points have been removed the optimization is run once more — this gives the final estimate. The covariance is reported as the approximated Hessian $J^T J$ where J is the Jacobian of the cost function.

4.4 Map Management

The map management module is responsible for collecting and combining various sensor inputs to manage the map and report the vehicle state information relative to the map. To construct the map it ties together several modules, including visual odometry, appearance based index, frame alignment and map estimation.

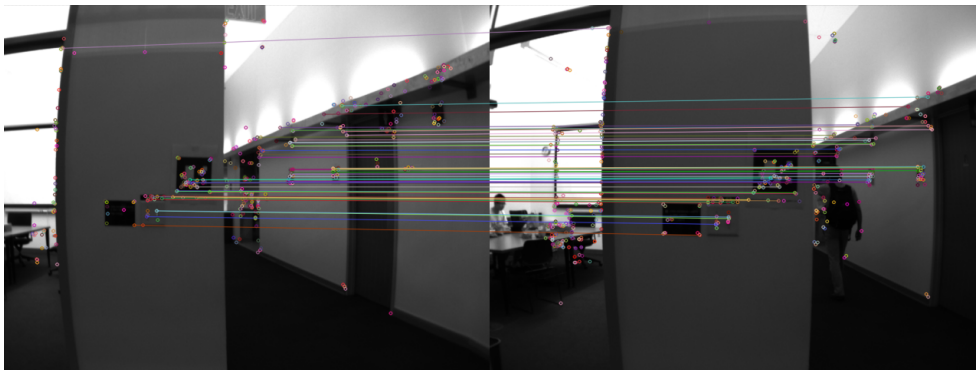
The primary inputs are frame matches from the visual odometry module. Each frame match contains the estimated motion between two frames which consists of the motion estimate and the uncertainty of the estimate. A status field indicates if the match succeeded or if not why it failed. If a match failed, the map module can use other estimates if available — e.g. wheel odometry or a motion model. The key steps in the mapping algorithms are:

- New match

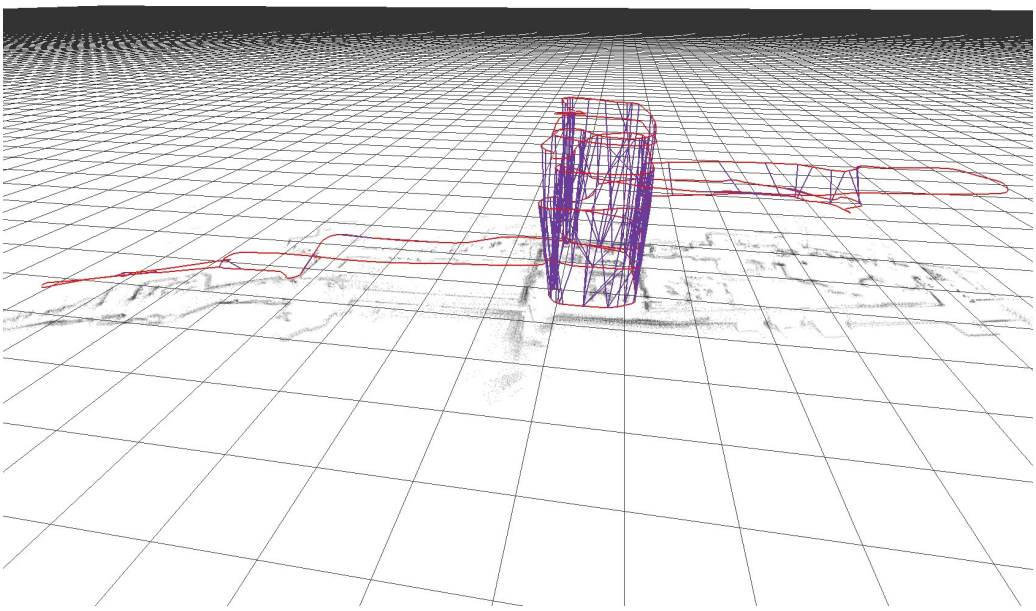


(a) Feature tracking

(b) An image from the stereo camera



(c) Loop closure



(d) Vision based pose graph with multiple passes

Figure 4-2: Examples of feature tracking for visual odometry and geometric loop closure verification.

- Check loop closure
- Add pose
- Chain measurements
- Incorporate IMU

The new matches come from the visual odometry and provide the transformation from the previous frame along with an uncertainty measure. The VO system issues these matches each time a keyframe is changed. Initially when the mapping system receives a new match it adds it to the transformation that links to the previous pose added to the map. If the camera moves a specific distance then a new node is added to the map and this chained estimate is used as a constraint between the two poses: the last one and the newly added pose. When the IMU information is used an additional constraint is added for the vertical component of the pose, i.e. it constrains the roll and pitch angles. In addition to tracking the sequential motion the system searches for potential loop closure to correct the drift of the estimate acquired by chaining the sequential motion. A high level description of the algorithm is given in Algorithm 1.

For any incoming frame a feature descriptor is computed for each keypoint in the new frame. The feature descriptors are maintained for later use in the appearance-based loop proposal and frame registration modules. Several different descriptor types are supported by our implementation including BRIEF, Calonder and SURF. In each case we utilize OpenCV to compute these descriptors.

Another responsibility of the map management module is to determine which poses to consider for loop closures. This includes local and global loop closures and is described in sections 4.5 and 4.6 respectively. The map module partitions the

Algorithm 1: Pose graph mapping algorithm

```
   Data: map is the map estimate
   Input: matches a queue for incoming matches
 1 foreach match ∈ matches do
 2     if match failed then
 3         use wheel odometry
 4     extract descriptor from current frame
 5     if distance from previous > threshold then
 6         add a new pose
 7         add transformation from previous pose
 8         check for a global loop closure
 9         if found then
10             add transformation to new pose
11     else
12         update transformation to last pose added
13 update map
```

space using a regular grid in 3 dimensions (x, y, and heading) and is referred to as a place map. The place map is used for tracking and later for the pose graph reduction presented in Chapter 5. This map is updated as new information is acquired and the pose graph changes. This partitioning can also be extended to take into account the elevation of the sensor and consider the view volume of each frame.

4.4.1 Visual SLAM Parameters

There are several parameters that need to be set for the visual SLAM system. Below we provide a description of these parameters.

Parameter	Typical value
Use WO	true
# inliers for WO fallback	15
Loop proposal threshold	0.2
Number of loop closures checked	10
# inliers to accept a loop closure	40

4.5 Active Node Registration

As the camera is moved around the environment it is possible to use the current position estimate to predict which frames overlap. If there are any nearby frames we try to align the current view to those frames. This is done by first finding a collection of putative matches by matching each keypoint in one frame to the keypoint that has the closest feature descriptor in the other frame. The matching is achieved using brute-force matching between the features in the two frames. Different types of feature descriptors can be used for the matching, including: BRIEF, SURF and Colander. The BRIEF descriptor is particularly useful because it is efficient both to compute and compare.

The initial putative matches will contain some incorrect correspondences. As with the visual odometry, we construct a consistency graph by considering the distance between a pair of points in the two frames. The inlier set is then determined by finding the maximal clique in the graph using the clique based inlier algorithm from FOVIS.

The initial estimate is computed by directly minimizing the squared error using Horn's algorithm (or Umeyama). Then the estimate is refined by minimizing the bi-directional reprojection error of the inlier points in each frame. This estimate is obtained using LM and a robust cost function. Points that have high reprojection

error are removed from the inlier set. The Hessian of the estimate is then used to estimate the covariance of the alignment. If a minimum number of inliers is found the constraint is added to the pose graph.

4.6 Global Node Registration

If registration to the active node fails, a global loop closure algorithm (across the entire set of poses) is used instead. A bag of visual words is used to describe each frame and an inverted index is used to efficiently search for similar frames. See [84] for more detail on this approach. Each match is scored and if the score is below a given threshold the frame is proposed as a possible loop closure. The loop proposal is then verified with a geometric consistency check, by registering the two frames, using the method described above.

When using SURF features the features are quantized using K-means. When binary descriptors such as BRIEF are used, a K-median is used instead. For the binary descriptors we used a library called "bbow" for the loop proposals [1]. Because we support multiple loop proposal modules we standardized the scoring to be in the range 0 to 1, where 1 would mean a perfect match. To evaluate the loop proposal method we ran the system with a low threshold on the proposal score, relying on the geometric consistency check to remove incorrect matches. We then looked at the distribution of the correct and incorrect matches as a function of the score. As seen in Figure 4-3 the loop proposal mechanism is not perfect, i.e. it does not separate the failed matches from the correct ones completely. So there is a trade off between choosing how many loop proposals to miss versus how much time to spend on the

[1]The library used for the binary descriptors is called Binary Bag-Of-Word (bbow) developed by Daniel Maturana — `http://dimatura.net/proj_bbow.html`

geometric verification.

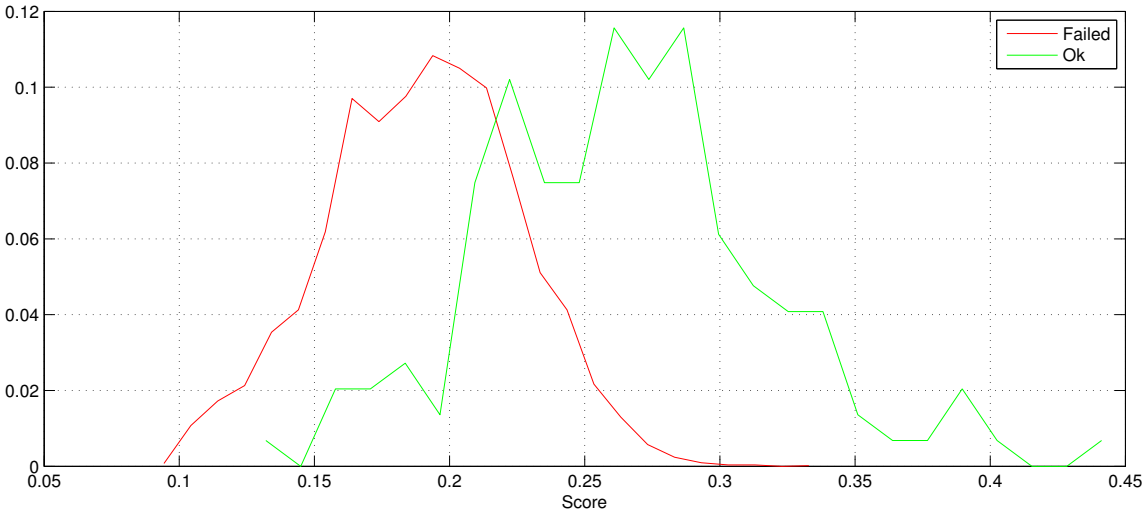


Figure 4-3: The distribution of the failed and successful loop proposals as a function of the score from the system.

4.7 Vertical Motion: Elevators

Many buildings contain multiple floors and it is reasonable to expect that the robot might be able to go from one floor to another, for example using an elevator. This type of motion is not observable by the vision system or the wheel odometry. Also it is not sufficient to rely on intent only, because the robot does not have full control over which floors the elevator will stop at. One could imagine using the loop proposal mechanism, but that might require the robot to exit the elevator.

One possibility is to use a barometer to track vertical motion. Instead, in this work we use the accelerometer sensor that is present in many robots, often as part of an IMU. Integrating the vertical accelerometer information over time provides us with the vertical displacement of the robot. The method is accurate because the

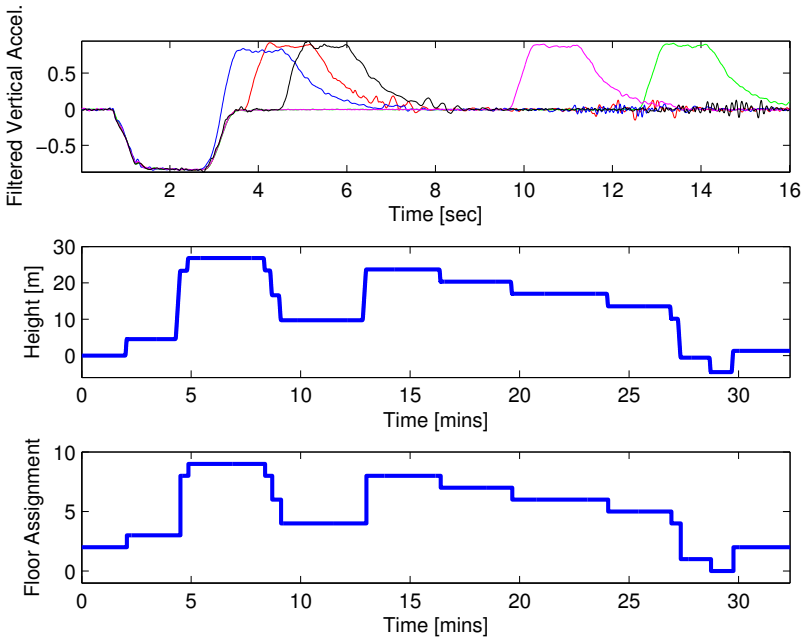


Figure 4-4: Top: Using the Z-component of the PR2's accelerometer, the start and end of elevator transitions can be detected using a matched filter. The figure illustrates 5 different elevator rides - showing that the acceleration is clearly repeated. Middle: By integrating this signal the elevation of each floor can be estimated. Bottom: During our 10 floor experiment (beginning and ending on Floor 3), the floor assignment can be determined using a simple lookup table. See Section 4.7 for more details.

Algorithm 2: Floor tracker

Data: $floors$
Input: **a** - acceleration, q - orientation

```
z_f ⟵ filter(z)
if z_f < threshold then
    motion up
else if z_f > threshold then
    motion down
else
    motion stop
update bias
if state == Stopped then
else if state == Decelerating then
    if motion == Stopped then
        state ⟵ Stopped
        update floor
else if state == Running then
position ⟵ position + velocity * dt
velocity ⟵ velocity + a * dt
if not stopped then
    time_tracking ⟵ time_tracking + dt
```

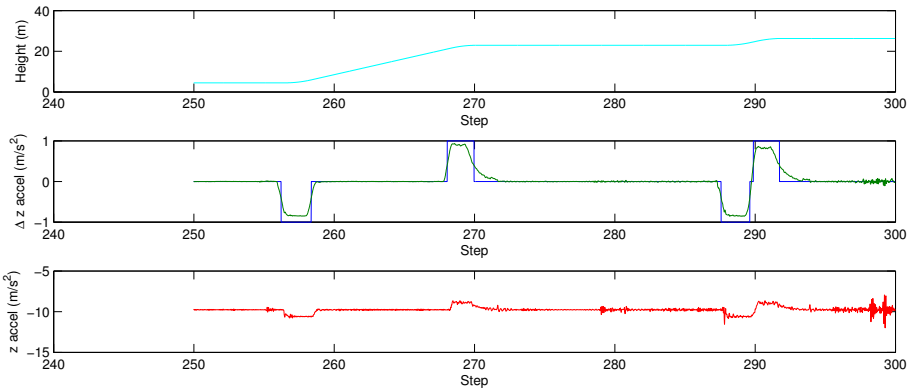


Figure 4-5: Top: The top shows estimated elevator height. Middle: The filtered acceleration with start and stop events marked with the blue line. Bottom: the raw acceleration input. See Section 4.7 for more details.

velocity at the start and end of the elevator transit are known to be zero. Results from our floor tracker are shown in Figures 4-4 and 4-5.

To assign these vertical displacements to floor numbers, a table of floor heights is maintained. Each time a transition is detected the table is searched for the closest floor. If the distance to that floor is within a given threshold it is accepted as the current floor, otherwise a new floor is added to the table. This could be further improved by treating each elevator ride as a measurement of the floor height and then use multiple rides to estimate all the floor heights.

Knowing the floor the robot is on is also useful for limiting loop closure search. Only matching to nodes on a single floor avoids wrong loop closures that can occur when different floors appear similar.

4.8 Results

To evaluate the visual SLAM system we collected several datasets from the 2nd floor of the MIT Stata Center. A PR2 robot was manually driven around the floor while the sensor data was recorded. The robot was equipped with a stereo camera, a Kinect, Microstrain IMU and Hokuyo laser range finder. Then laser scans from the dataset were manually aligned to floor plans to generate ground truth poses for the dataset. The error metric used was the absolute distance between the estimated pose and the groundtruth pose. In addition to the accuracy evaluation we also considered the performance of the different modules in the system.

4.8.1 Visual Odometry

First we determine the accuracy of the VO and compare it to the wheel odometry. The accuracy was evaluated by having the visual SLAM run using only dead reckoning. Then the trajectories were fixed at the first pose of the groundtruth. As shown in Figure 4-6 the position estimates from the integrated VO and wheel odometry (WO) drift as the vehicle moves from its starting position. On this sequence the VO performs better, even though alignment failed several locations. Figure 4-7 shows that both the position and the heading accuracy of the VO outperforms the wheel odometry. The error in heading is the largest contributor to the absolute position error when using the wheel odometry. At some point the wheels might have slipped giving an incorrect measurement. These results suggest that using the VO as the primary motion input is the best choice. In the next section we will look at how these choices affect accuracy when running the full visual SLAM system.

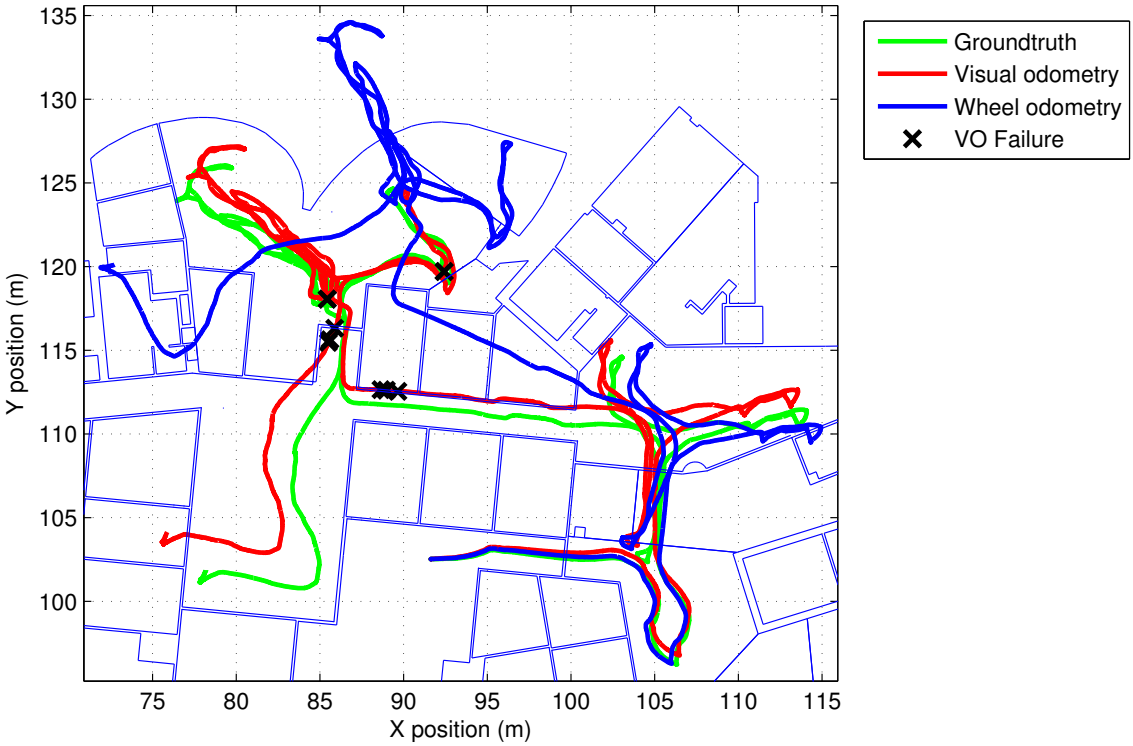


Figure 4-6: The dead reckoning trajectories for visual odometry and wheel odometry are shown in comparison to the ground truth trajectory.

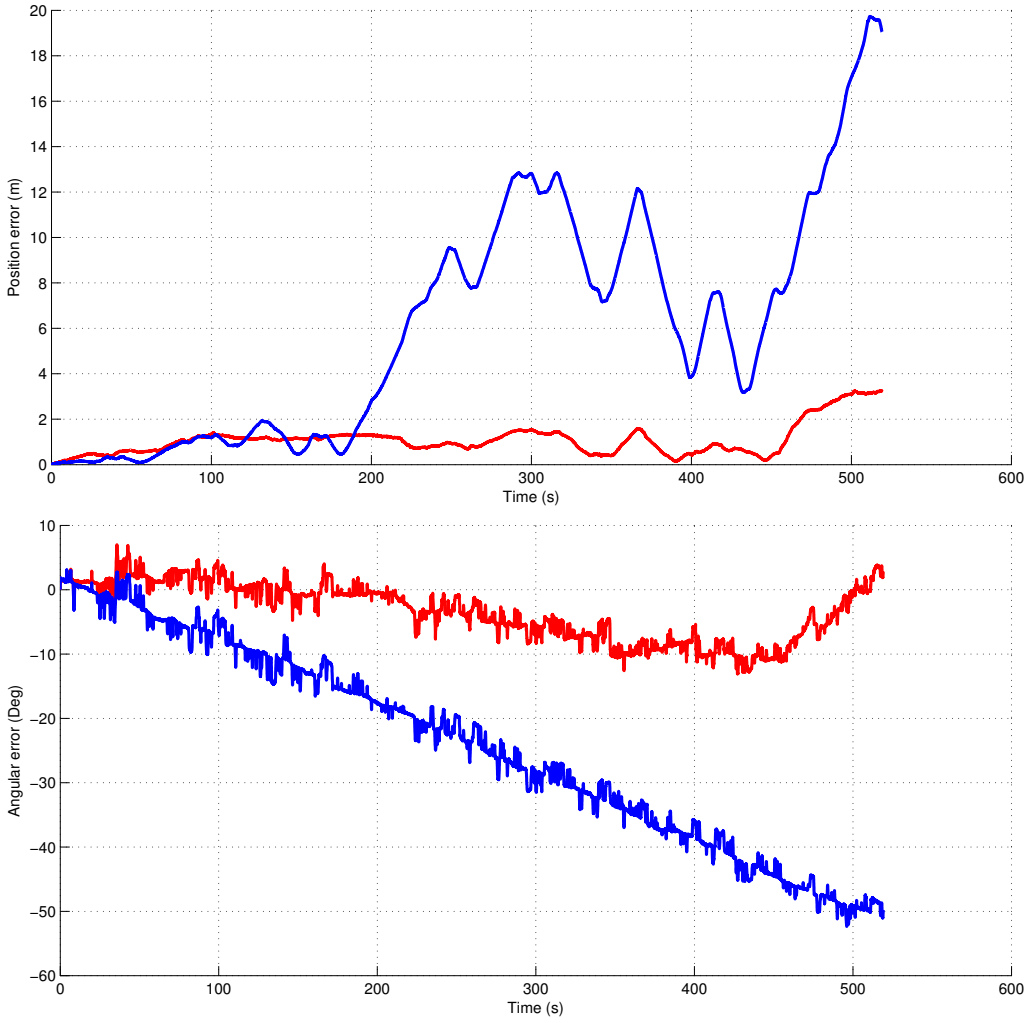


Figure 4-7: Comparing drift in visual odometry and wheel odometry.

4.8.2 Accuracy of the Visual SLAM System

In this section the accuracy of the visual SLAM system is evaluated. The effects of solely using vision are compared to the use of wheel odometry for error recovery and using an IMU to bound the drift of the roll and pitch of the camera. As shown in Table 4.2 combining all the sensors improves the performance of the system. Using the VO as the sequential motion estimate performs better then using the WO as was expected from the results in the previous section. The different trajectories estimated are shown in Figure 4-8.

Incorporating the IMU has a significant effect on bounding the drift along the vertical axis. When using only VO, the height error is approximately 1m towards the edges of the map while it is closer to 30cm when the IMU information is used. This is shown in Figure 4-9 where the poses are plotted along the z and x axis of the map.

Another thing to look at are the error distributions which are shown in Figure 4-10. The errors are computed as the absolute distance to the groundtruth pose. There are some outliers in the distribution. These tend to be towards edges of the maps where they are not as well constrained as the well connected parts in the center of the maps.

	Median (m)	Mean (m)	StdDev (m)
Vision only	0.74	0.73	0.55
WO + IMU + Vision for loop closure	0.71	0.79	0.52
Vision + IMU + WO	0.59	0.58	0.38

Table 4.2: Accuracy for different variants of the SLAM algorithm. This is evaluated on the Stata 2nd 01-25-12-1 dataset using the stereo cameras.

Another run from the Stata Center 2nd floor was evaluated, and the mean error was 0.49m, the results are shown in Table 4.3.

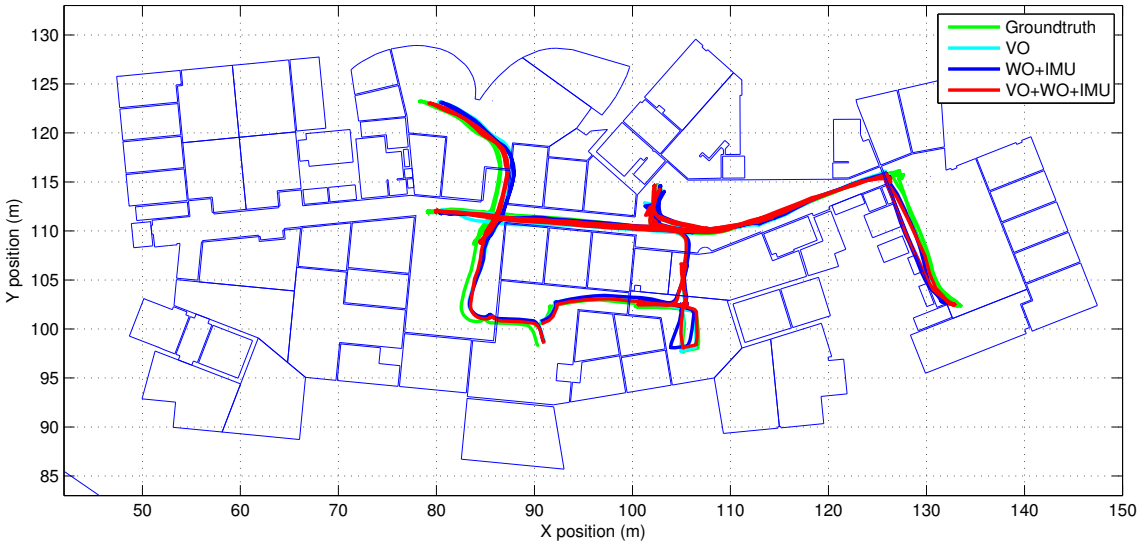


Figure 4-8: Comparison between pose graph accuracy using different combination of sensors.

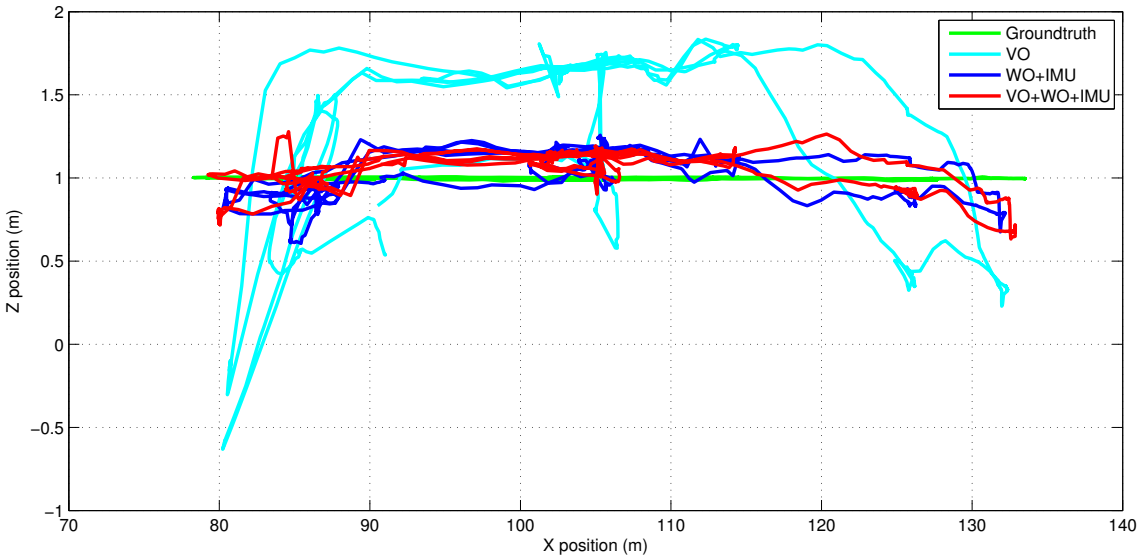


Figure 4-9: Comparison between pose graph accuracy using different combination of sensors.

	Median (m)	Mean (m)	StdDev (m)	Duration (min)
Vision + IMU + WO	0.40	0.49	0.28	36

Table 4.3: Accuracy for Visual SLAM using a stereo camera. This is evaluated on the Stata 2nd floor 2012 01-18-09 dataset.

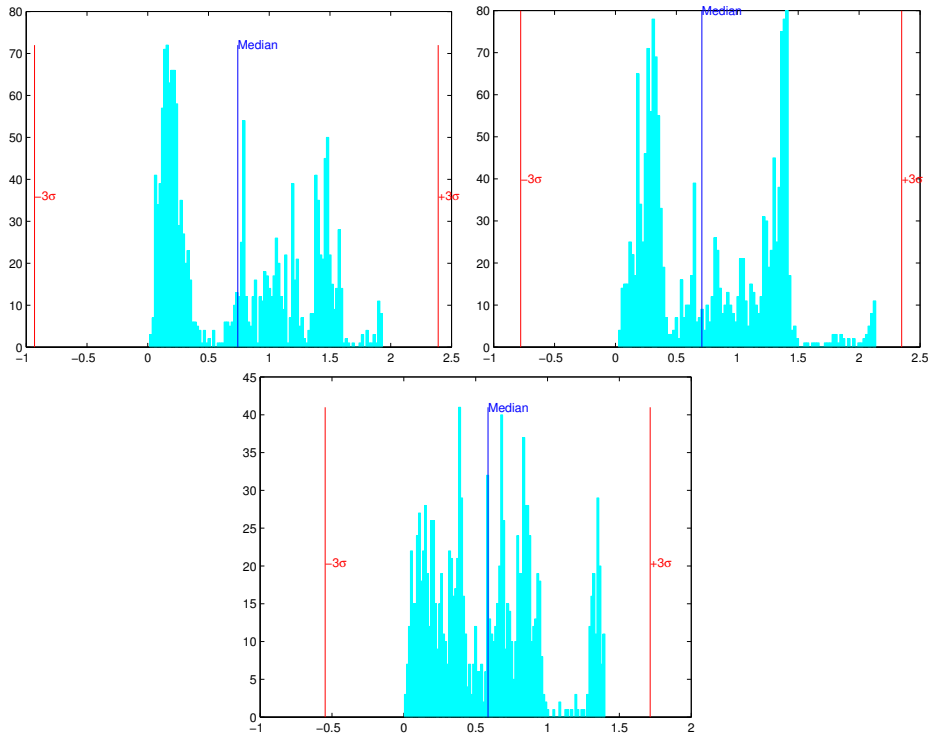


Figure 4-10: Shows the error distribution for a pose graph when using different combinations of sensors. The top left is for VO only, top right is WO + IMU, and bottom is VO + IMU + WO for recovery in case the VO fails.

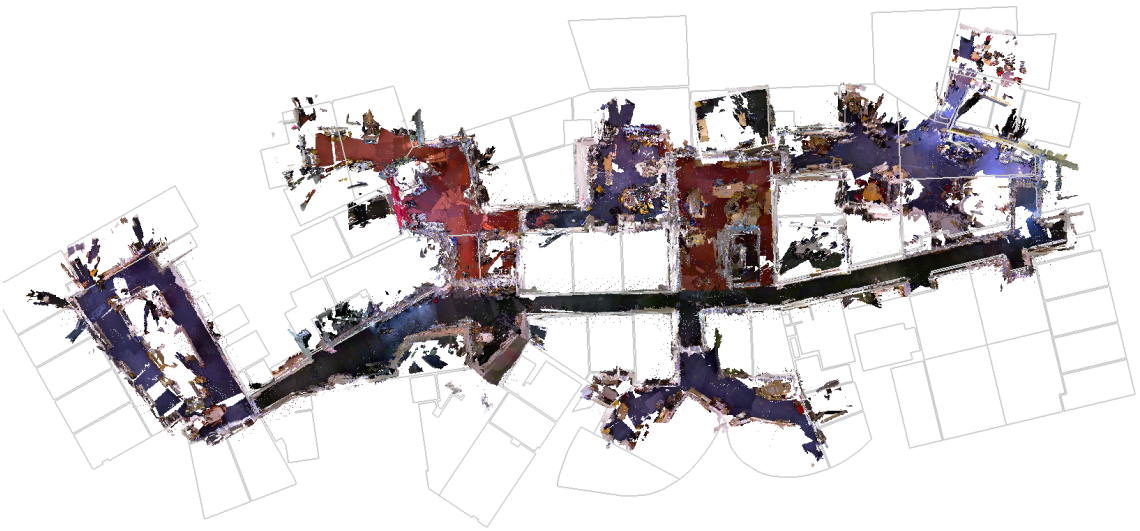


Figure 4-11: Map created using a Kinect camera. The map is aligned using the best similarity transform that fits the pose graph to the ground truth.

For a qualitative comparison we processed a sequence from the Kinect camera. Then each point cloud associated with the poses in the map was reprojected to create a colored 3D map. The results are shown in Figure 4-11. It was aligned to the floor plan by finding a transformation that minimized the distance of the poses in the map and the position provided by the groundtruth results. The width of the floor is around 90m. Globally the map aligns fairly well with the building floorplan.

4.8.3 Performance of the System

In addition to evaluating the accuracy of the system, we also looked into the performance of the system, which is important for on-line applications. To stress the system we used a dataset that consists of 4 hours of stereo data from the PR2. The timing results are shown in Figure 4-12. The vertical axis is the percentage of computation time used. As more and more nodes are added to the map the CPU

eventually saturates and most of the time is spent on optimizing the graph. The appearance based search is another component where the computational complexity grows as the number of nodes increases.

This illustrates the main problem with the pose graph formulation. When operating for long duration the pose graph grows without bounds, even though the size of the environment is fixed. So it is necessary to reduce the graph in some way to bound the computational complexity. In the next chapter we will address this problem and propose a reduced pose graph algorithm that can efficiently process this sequence and generate a map of comparable accuracy.

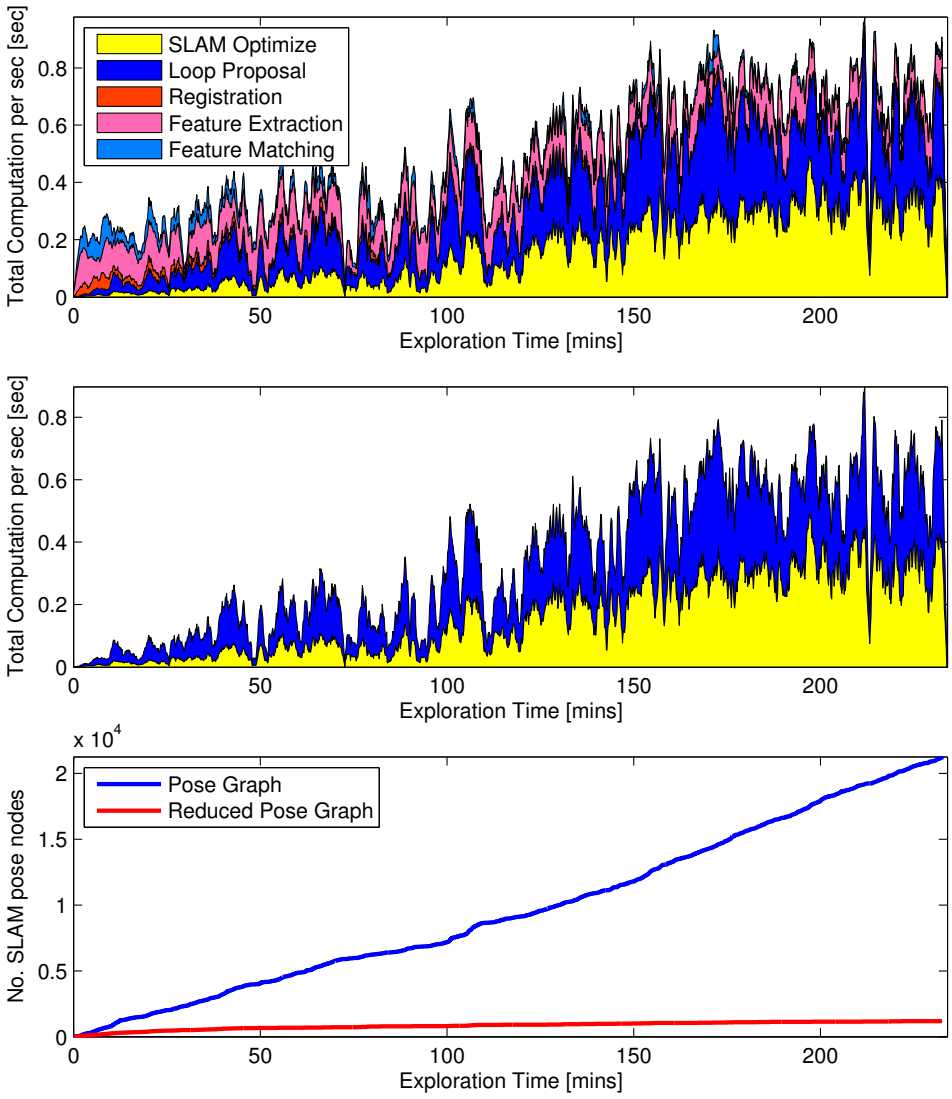
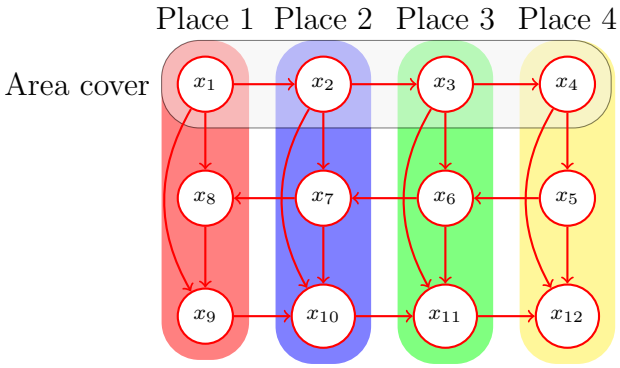


Figure 4-12: Timing results when using a **full pose graph** for a 4 hour sequence. The top plot shows the time for each component of the SLAM system as a function of exploration time. The middle plot shows the time for those components that have growing time complexity. The bottom plot shows the number of nodes in the graph, both for the pose graph and the reduced pose graph.

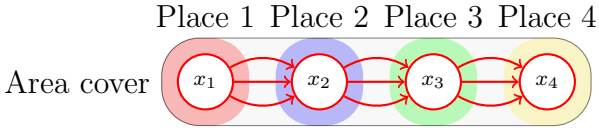
Chapter 5

Reduced Pose Graph for Temporally Scalable Visual SLAM

In the previous chapter a pose graph based visual SLAM system was presented. One of the limitations encountered with that algorithm is that as the robot continuously navigates through the environment more poses are added and the graph grows larger with time. As the graph grows larger it will eventually affect the performance of the system as more and more of the time is spent updating the map estimate and eventually running out of memory as more poses are stored. In this chapter we present an approach that constructs and maintains a map such that the complexity is significantly reduced compared to the full pose graph — we call this method the reduced pose graph (RPG). The core concept, as demonstrated in Figure 5-1, is to construct a minimal representation of the environment yet continually integrate new information for improved robustness and accuracy. We will show how the complexity is drastically reduced for long term operations and how the accuracy improves over time. The evaluation is performed using a multi-floor long-term dataset collected



(a) Traditional pose graph



(b) Our reduced pose graph

Figure 5-1: Comparing our reduced pose graph with the full pose graph on a small example. The same environment is traversed three times. The area is covered by the first four poses. The reduced pose graph reuses these poses and transforms new constraints into constraints between the existing poses.

with the PR2 robot.

5.1 Introduction

The motivation for the reduced pose graph is to retain a sufficient amount of poses for representing the explored area. The concept of pose graph reduction to achieve temporal scalability is intuitively appealing, and indeed has been proposed in the previous literature in several contexts [36, 106, 107]. Pose graph reduction is related to the use of keyframes in PTAM [57], but the use of keyframes alone presents several shortcomings which we seek to avoid in our approach. In particular, using new information only to track the robot/camera pose results in loss of information

vs. our approach which continues to make use of new measurement information to improve the map over time.

5.2 Graph Reduction

As defined in Chapter 2 a pose graph consists of N pose nodes $X = \{x_i\}_{i=1}^{N}$ and M constraints $Z = \{z_k\}_{k=1}^{M}$. The main problem with the standard pose graph representation is that the state space continuously grows as more poses are added.

One approach to reducing the graph is to marginalize out some of the nodes to reduce the state. We formulate the problem with the joint probability density over the poses $p(X, Z)$. Now let $x_r \in X$ be a node that should be removed and $X = \{X_0\ X_N\ x_r\}$ where X_N are variables that connect to the same factors as x_r

$$
\begin{aligned}
p(X_0, X_n, Z) &= \int_{x_r} p(X_0, X_N, x_r, Z) \mathrm{d}x_r && (5.1)\\
&= \int_{x_r} p(X_0, Z|X_N) p(x_r, Z|X_N) p(X_N) \mathrm{d}x_r && (5.2)\\
&= p(X_0, Z|X_N) \int_{x_r} p(x_r, Z|X_N) p(X_N) \mathrm{d}x_r && (5.3)\\
&= p(X_0, Z|X_N) \int_{x_r} p(x_r, X_N, Z) \mathrm{d}x_r && (5.4)
\end{aligned}
$$

and by linearizing around the current mean making an assumption of Gaussian errors then $p(x_r, X_N, Z)$ is a Gaussian and x_r can be marginalized out. This causes all the variables X_N to become connected so the graph loses sparsity.

Kretzschmar et al. [63] address this by approximating the marginal distribution with another distribution that factorizes into $p(x_i|x_j)$ terms, and then uses Chow-Liu algorithm to find such a distribution that minimizes the KullbackLeibler (KL)

divergence between the two.

Another approach is taken by Eade et al. [17] where the marginal distribution is approximated by using the existing measurements to construct new connections between all the nodes that are neighbors to the node to be removed. Instead of reducing the marginal distribution during construction, edge pruning is used. When the degree of a node exceeds a threshold some of the edges are removed to bring the degree of the node below the set threshold. The edge with the least residual error is removed given that it does not disconnect the graph. That is ensured by checking the distance between the nodes on either side of an edge. If the distance is greater then a given threshold the edge is not removed. In addition, if there is already an edge between two nodes then those edges are combined.

In contrast to the methods above, our approach discards sequential constraints to avoid the marginalization causing increased density of the graph. Additionally, this is an efficient approach that does not require modification of the existing graph. This algorithm is described in detail in following section.

5.3 Reduced Pose Graph

In this section we describe the reduced pose graph algorithm. In Figure 5-2 we illustrate the reduction technique by comparing the construction of a full pose graph and a reduced pose graph in the top and bottom figure respectively. The blue dart indicates the current position of the robot. In step (1) a single loop closure to the map has been acquired – as indicated by the blue edge. In step (2) a node is added to the graph so the loop closure can be used and a loop closure to x_j is detected. In step (3) a node has been added so the last loop closure can be incorporated into the graph. No loop closure is considered at the current pose because the robot has not

moved far enough from the previous place. Finally in step (4) the robot has traveled enough distance that a new pose is added and the whole process is repeated.

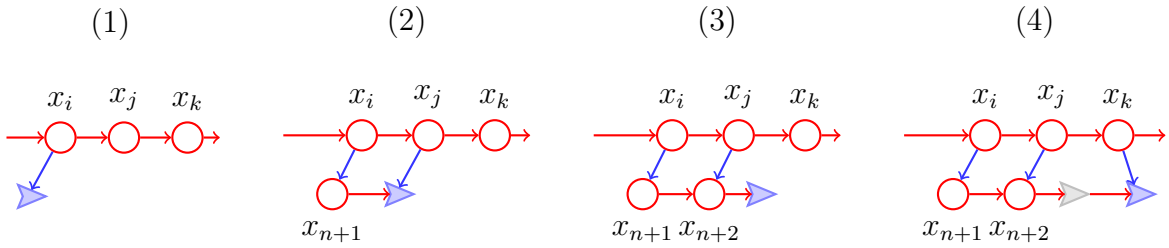
In contrast the reduced pose graph algorithm will estimate the transformation from pose x_i to x_j. So in step (2) the transformation to x_i is updated instead of adding a new node to graph. In step (3) a complete chain from x_i to x_j (shown in green) has been formed and can now be added as a constraint to the graph. Furthermore the robot continuously tracks its position relative to the active node x_j. Then in step (4) the process is repeated as other places are visited.

The reduced pose graph consists of N pose nodes $X = \{x_i\}_{i=1}^{N}$ and M constraints $Z = \{z_k\}_{k=1}^{M}$. A pose node contains the actual pose as well as the sensor measurements required to construct a map and to recognize when the pose is revisited. A constraint z_k measures the spatial relationship between two poses i_k and j_k. Because the measurements are noisy we consider the probability distribution $p(X, Z)$, in particular we are interested in the maximum likelihood solution of $p(X|Z)$ were X are our parameters of interest. By design, the joint distribution factorizes as follows

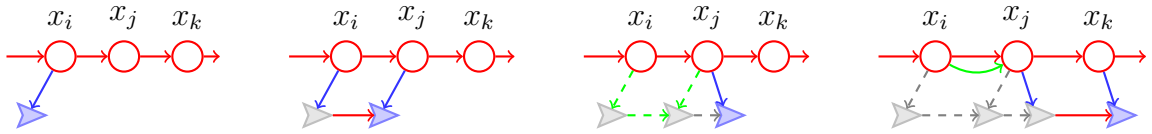
$$p(X, Z) = \prod_{k=1}^{M} p(z_k|X_k) \prod_{i=1}^{N} p(x_i), \tag{5.5}$$

where X_k is some subset of X. Then as mentioned in Section 2.4.2 we can compute the maximum likelihood estimator X^* by

$$X^* = \underset{X}{\operatorname{argmin}} \sum_{k} C_k \left(f(x_{i_k}, x_{j_k}) - z_k \right), \tag{5.6}$$

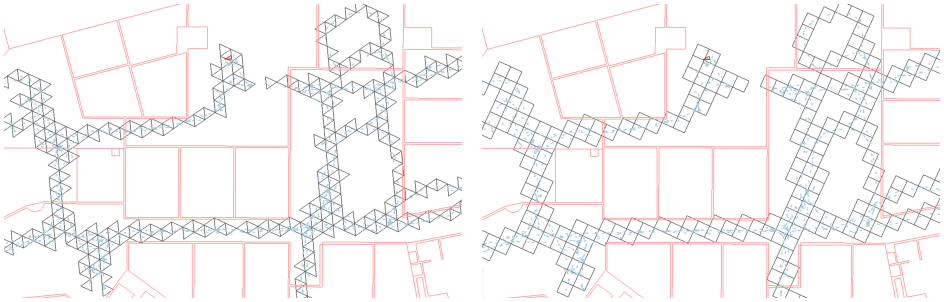


(a) Full pose graph

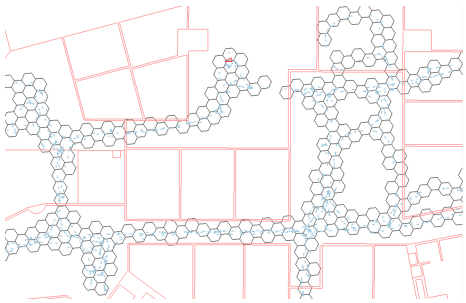


(b) Reduced pose graph

Figure 5-2: A comparison of the construction of a full vs. reduced pose graph. The red circles are arbitrary poses in the map that the robot passes. The blue darts are poses along the vehicle trajectory. **Full pose graph**: (a) A single loop closure is detected. (b) To add it to the map a new pose (x_{n+1}) has to be included in the map. (c) Again a loop closure is detected and a pose (x_{n+2}). (d) The same procedure is repeated for each loop closure and another pose (x_{n+3}) is added. The result is that several poses are added to the map that are not added to the reduced pose graph. **Reduced pose graph**: (a) A single loop closure has been computed between the current pose and map pose x_i. (b) The vehicle continues moving and acquires a new loop closure to map pose x_j. (c) Another loop closure is detected. Now the chain of constraints (green links) can be compounded into a single constraint. The sequential constraint between the last two poses is dropped. (d) The new constraint (green link) has been added and the vehicle has continued to the next pose and is ready to create a constraint between x_j and x_k.



(a) Triangular grid (b) Square grid



(c) Hexagonal grid

Figure 5-3: Partitioning strategies

5.3.1 Partitioning

To decide which poses to retain, the environment needs to be partitioned in some way. Which type of partitioning scheme is used depends on the environment and the motion of the robot, e.g. for a wheeled robot in an office environment one of the three regular tessellations: equilateral triangles (Figure 5-3(a)), squares (Figure 5-3(b)) or hexagons (Figure 5-3(c)) can be used.

It is interesting to note that the hexagonal partitioning shows up when studying neurons related to navigation in rats. A collection of cells in the hippocampus, called grid cells [38], are periodically activated on locations that are centered on the vertices of an equilateral triangular lattice. These vertices then coincide with the center points of hexagons that tessellate the space.

Alternatively the partitioning could be implicitly based on the features seen in each view. If a view sees a feature that is not visible in other frames, or only visible in a few other frames then it should be added. In that case the selection of nodes should be a set of views that see all the features that have been detected. A variant of this approach is to consider the view volume of each view, then we find a set of poses that cover the workspace. In our work we typically used a 1.5m square grid and the heading was split in 30 degree intervals.

5.3.2 Graph Construction

Let us now consider how the reduced pose graph is constructed; a summary is provided in Algorithm 4. The graph is initialized by defining the first pose as the origin. There are three primary operations performed: tracking, adding nodes, and adding loop closures. Let x_a denote the active pose and Δ_t^a denote the transformation from the active node to the current position at time t. For simplicity, in the figures below we denote the sequential motion estimates as u_i though they could come from either visual odometry or wheel odometry. And the v_i are intermediate vehicle poses from the last active pose and are not a part of the map.

Tracking

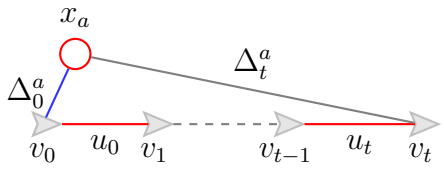


Figure 5-4: A graphical model demonstrating the tracking phase.

The tracker estimates the distribution $p(\Delta_t^a|U_t, Z_t)$ where U_t and Z_t are the odometry and visual constraints respectively. This is illustrated in Figure 5-4. Based on our Markov assumption the distribution can be computed recursively

$$p(\Delta_t^a|U_t, Z_t) \propto p(\Delta_t^a|u_t, z_t, \Delta_{t-1}^a)p(\Delta_{t-1}^a|U_{t-1}, Z_{t-1}). \tag{5.7}$$

In practice we track the location distribution by compounding with the new measurement at each stage. The compounding of uncertain transformations follows the notation of [94]. Given a chain of transformations $\{z_{12}, z_{23}\}$ with covariances $\{\Sigma_{12}, \Sigma_{23}\}$ respectively, the compounded transformation z_{13} is computed as follows

$$z_{13} = z_{12} \oplus z_{23} \tag{5.8}$$

$$\Sigma_{13} = J_{1\oplus}\Sigma_{12}J_{1\oplus}^T + J_{2\oplus}\Sigma_{23}J_{2\oplus}^T, \tag{5.9}$$

where $J_{1\oplus}$ and $J_{2\oplus}$ are the Jacobians of the compound operation $\oplus$ with respect to z_{12} and z_{23} respectively. This is the first order approximation of the mean and covariances, where z_{12} and z_{23} are assumed to be independent. The factor added to the graph will then be $|f(x_1, x_3) - z_{13}|_{\Sigma_{13}}$, where f is a function that computes the transformation between x_1 and x_3, i.e. $x_3 = x_1 \oplus f(x_1, x_3)$.

Adding Nodes

When $x_a \oplus \Delta_t^a$ is in a location that contains no existing pose, then a new node x_{N+1} is added to the graph and the factor graph is updated according to

$$p(X_{N+1}, Z_{M+1}) \propto p(z_{M+1}|x_a, \Delta_t^a)p(X_N, Z_M). \tag{5.10}$$

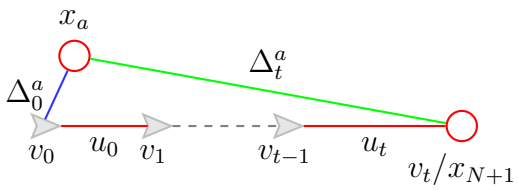


Figure 5-5: A graphical model demonstrating when a new node is added to the map.

by adding the transformation Δ_t^a as a constraint to the new node. The newly added node becomes the new active node x_a and the tracker is re-initialized.

Add Loop Closure

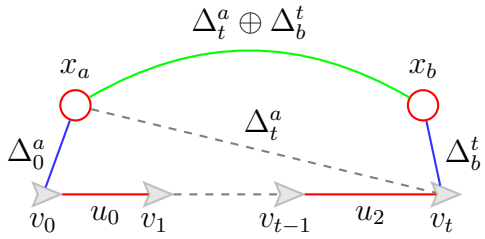


Figure 5-6: A graphical model demonstrating when a new loop closure is added.

The tracker is always trying to find an alignment with an existing pose. First the node that is closest to the current position is tested. If that fails a global appearance based search is used to discover a loop closure. If successful, a new constraint is added to the graph. The constraint is given by

$$p(\Delta_a^b | x_a, \Delta_t^a, \Delta_b^t, x_b), \tag{5.11}$$

where x_b is the node being registered to. After the loop closure is added x_b becomes the active node. Next the robot needs to re-localize relative to the map to avoid

reusing the previous measurement. A re-localization will initialize the active node and the relative position to it. Alternatively, if enough inliers were found in the loop closure, they could be split into two sets, one for the loop closure and the other for re-localizing. In comparison a full pose graph approach would have added a new pose at the point of loop closure.

In case where x_a and x_b are the same node it is possible to choose if the current estimate or this loop closure should be used as the estimate of the position relative to x_a by considering which has higher entropy, using the test $det(\Sigma_t^a) > det(\Sigma_t^b)$.

Algorithm 3: Motion tracker algorithm

Data: Q a message queue for the mapper
Input: $frame$ is a new camera frame
compute motion estimate
publish new position
if *keyframe changed* **then**
    add previous match to mapper
update match

The sparsity of the graph is enforced by discarding some of the sequential constraints. Lets consider an example to better understand what information is lost. Let x_a, x_b and x_c be poses in the map and v_i for $i = 0...1$ be vehicle poses that have not been added to the map. In addition we have loop closure measurements z_a, z_{b1}, z_{b2}, z_c and sequential constraints u_i for $i = 1..3$. The probability density we are interested in is

$$p(x_a, x_b, x_c, v_1, v_2, v_3|Z, U) \propto p(z_a|x_a, z_1)p(u_1|v_0, v_1)p(z_{b1}|v_1, x_b) \tag{5.12}$$

$$p(u_2|v_1, v_2) \tag{5.13}$$

$$p(z_{b2}|x_b, v_2)p(u_3|v_2, v_3)p(z_c|x_c, v_3) \tag{5.14}$$

Algorithm 4: Reduced Pose Graph Mapping

```
Data: map the map estimate
Input: matches a queue for incoming matches
foreach match ∈ matches do
    if match successful then
        update Δ_t^a using match
    else
        update Δ_t^a using wheel odometry
    extract descriptors from current frame
    check for loop closure to active node
    if not found then
        check for a global loop closure
    if found then
        if found node is active node then
            update transform to active node
        else
            add transformation to found node
            set found node as active node
    else
        if in a new place then
            add a node for new place
            set new node as active node
update map
```

The dependency between the variables is illustrated with the graphical model in Figure 5-7.

Now if the vehicle poses $v_{0:3}$ are marginalized out, then we end up with a factor that connects all the poses $x_{a:b}$ which in general does not factorize. In the Gaussian case we have $O(k^2)$ factors where k is the number of nodes in the factor. On the other hand, if we discard the measurement u_2 then the factor $p(u_2|v_1, v_2)$ drops out

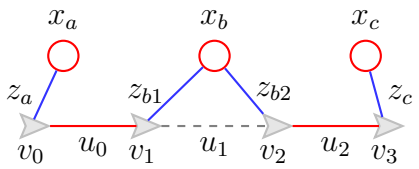


Figure 5-7: A demonstration of what information is discarded by the reduction algorithm.

and the marginal distribution factorizes into two factors as follows

$$p(x_a, x_b, x_c|Z, U) \propto \iiiint p(x_a, x_b, x_c, v_1, v_2, v_3|Z, U)\mathrm{d}v_0\mathrm{d}v_1\mathrm{d}v_2\mathrm{d}v_3 \tag{5.15}$$

$$\begin{aligned} &= \left(\iint p(z_a|x_a, z_1)p(u_1|v_0, v_1)p(z_{b1}|v_1, x_b)\mathrm{d}v_0\mathrm{d}v_1\right) \\ &\quad \left(\iint p(z_{b2}|x_b, v_2)p(u_3|v2, v3)p(z_c|x_c, v_3))\mathrm{d}v_2\mathrm{d}v_3\right) \end{aligned} \tag{5.16}$$

$$= f_1(x_a, x_b)f_2(x_b, x_c) \tag{5.17}$$

where the factors f_1 and f_2 are exactly the transformations estimated by the tracking and loop closure part shown before. In the result section we demonstrate that even if we discard this information the resulting map has comparable accuracy to the full pose graph which uses all the measurements.

In the case of the graph pruning presented by Kretzschmar et al. [63] the information discarded will actually depend on which node gets selected for removal. If v_1 is removed after v_2 has been added, then in the linearized problem there will be links between v_0, v_2 and x_b and the links preserved are the most informative links, resulting from the Chow-Liu algorithm. If x_b is removed then some of the information that is already in the map might be discarded. How much information is removed will depend on the connectivity of x_b. As the system progresses and more edges merge

during the marginalization it gets difficult to keep track of what information gets removed.

For the edge pruning algorithm proposed by Eade et al. [17] the cliques are always constructed and edges are only removed when the degree of a node reaches some threshold. So the odometery constraint u_1, that the RPG discards, might get discarded at some point if it belongs to an edge with the least residual error of its neighbors. In addition, the marginalization procedure used will reuse measurements which can make the estimator inconsistent.

In addition to the benefit of RPG reducing the dimensionality it is also beneficial that most of the updates to the map are adding new constraints and not variables, which can be applied incrementally in an efficient manner using the iSAM algorithm [53] for pose graph optimization, which also includes an incremental implementation of the Powell's Dog Leg algorithm [89] allowing the use of robust cost functions for the edges.

5.4 Results

To evaluate the RPG algorithm we used the groundtruth vision dataset that was mentioned in Chapter 4. The RPG is compared to a full pose graph and the reduction algorithm presented by Eade et al. We present results for a 4 hour dataset collected on the same floor, so the same area is repeatedly explored. In addition, we also show results from processing a 9 hour sequence covering multiple floors in the MIT Stata Center. The data was collected over a 6 month period. For the timing information a Intel(R) Core(TM) i7-2920XM CPU @ 2.50GHz machine was used. The system runs in two separate threads so two cores were used, one for the visual odometry and another for the mapping.

5.4.1 Comparison to a Full Pose Graph

To evaluate the system we considered both the accuracy of the map and the computation time used. This was compared using a 4 hour dataset collected on the second floor of the Stata Center. The data was collected on a single floor and includes multiple visits to the same location. After the 4 hours the full pose graph approach used the majority of the available computation time for optimizing the graph (See Figure 5-8). While the reduced pose graph (RPG) was able to comfortably process a longer sequence, as shown in Figure 5-16. For the full pose graph as more time is spent on the map estimation it will eventually affect the localization accuracy of the system. The core system infrastructure was developed to adaptively balance the load so that it can meet real-time requirements when running on a robot. This is achieved by having the mapping thread consider a limited number of loop closure proposals and also adapting how often the map optimization is executed. Thus, as the pose graph grows, the system cannot keep up, and fewer and fewer frames are registered with the map.

The resulting maps for this sequence are shown in Figure 5-9(a) and Figure 5-9(b) for the full pose graph and reduced pose graph respectively. The accuracy of the estimated maps is provided below and the distribution of the errors is shown in Figure 5-10.

	Median (m)	Mean (m)	StdDev (m)	Poses
Full pose graph	0.37	0.43	0.27	28520
Reduced pose graph	0.44	0.46	0.29	1363

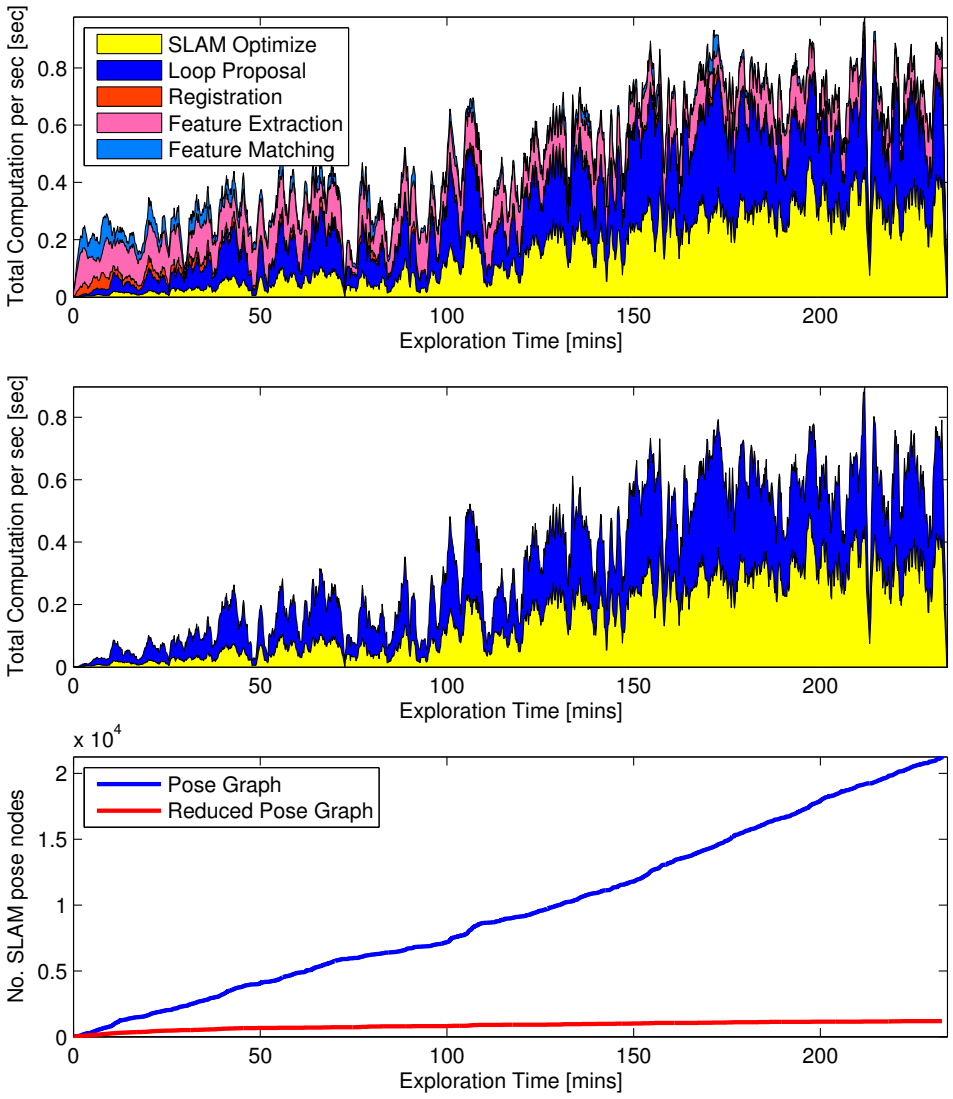
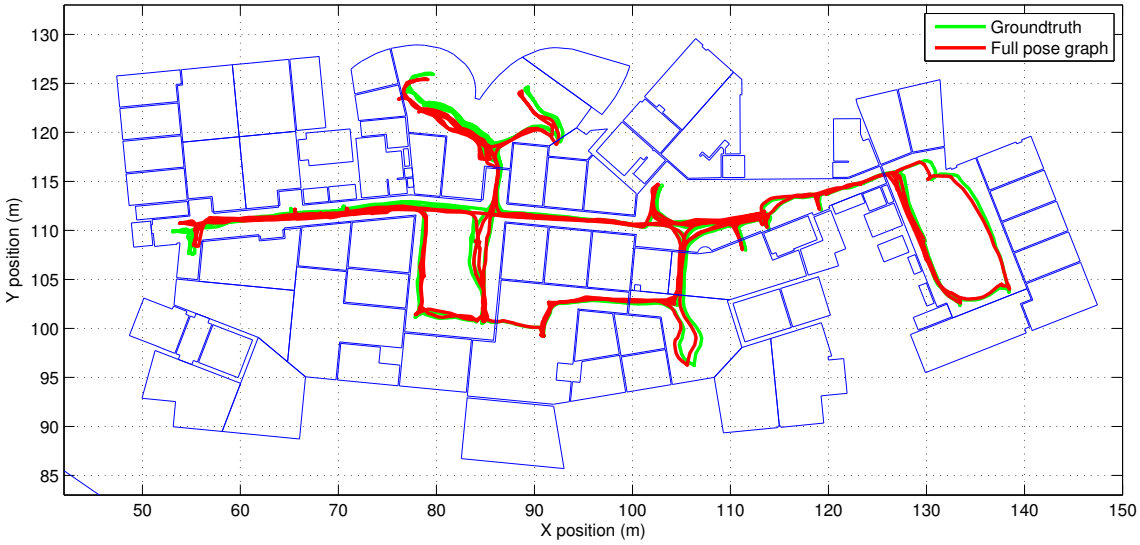
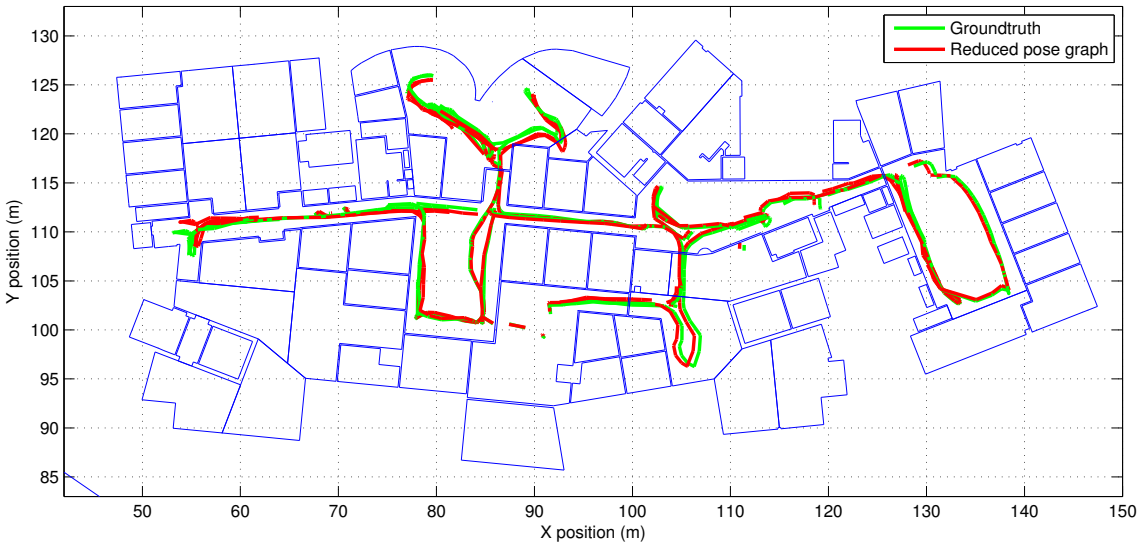


Figure 5-8: Timing results when using a **full pose graph** for a 4 hour sequence. The top plot shows the time for each component of the SLAM system as a function of exploration time. The middle plot shows the time for those components that have time complexity that grows with the size of the graph. The bottom plot shows the number of nodes in the graph, both for the pose graph and the reduced pose graph.



(a) Pose graph – 28520 poses



(b) Reduced pose graph – 1363 poses

Figure 5-9: A comparison of a full pose graph vs. a reduced pose graph from 4 hours of traversal. The green is the groundtruth trajectory and red are the estimated trajectories. The average error for the full pose graph is 0.43m while it is 0.47m for the reduced pose graph.

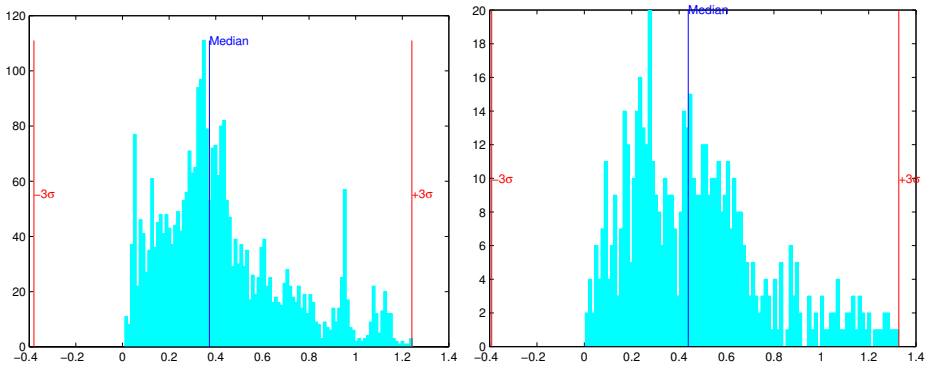
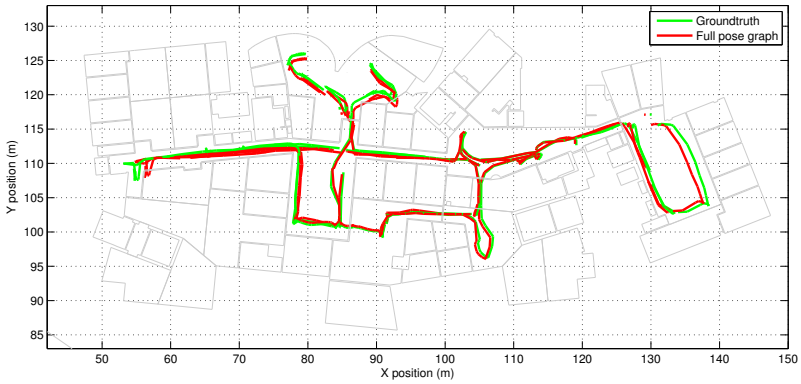


Figure 5-10: Shows the error distribution for the pose graph and reduced pose graph.

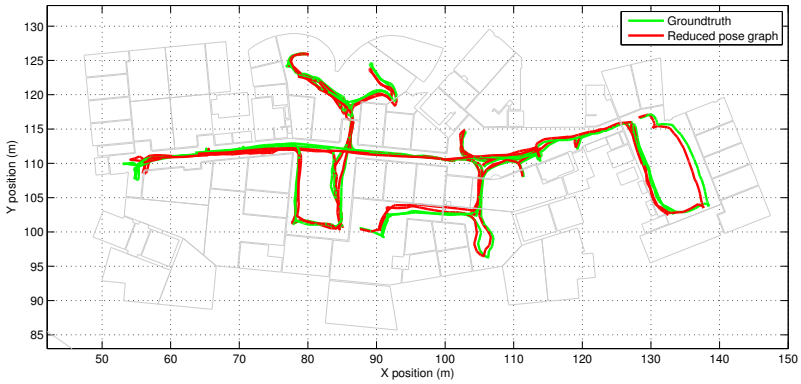
5.4.2 Comparison to Edge Pruning

We implemented the edge pruning algorithm for comparison to the reduced pose graph. As our system uses stereo vision we did not implement the complete system as described. Instead we adapted it to our visual SLAM system. The three primary operations, edge combination, marginalization and edge pruning were implemented in the same way. The degree threshold used was 8 as in the original paper. The max length threshold was set to 10, but it was not specified in the paper. The same partitioning as used in the reduced pose graph was used to choose which nodes to remove. The initial pose is kept and others are removed. In addition we never remove the most recent poses. The median errors are similar though the edge pruning contained some outlier poses. The methods used in the edge pruning algorithm can also be applied on top of the reduced pose graph to further bound the size of the graph.

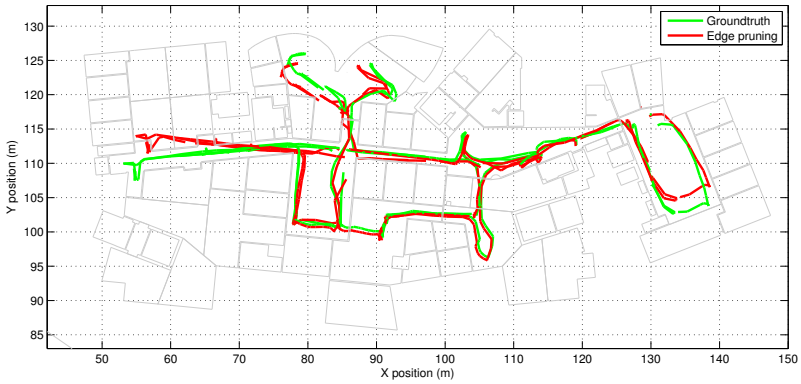
	Median	Mean	StdDev	Poses
Full pose graph	0.34	0.471	0.34	4062
Reduced pose graph	0.42	0.52	0.34	994
Edge pruning	0.57	0.93	0.89	607



(a) Pose graph – 4062 poses



(b) Reduced pose graph – 994 poses



(c) Reduction by node removal – 607 poses

Figure 5-11: Comparison of a full pose graph, a reduced pose graph and edge pruning.

5.4.3 Error Over Time

One question to consider is if it helps to continually add information to the graph. By continually adding measurements, the overall accuracy of the map can be improved over time, as shown in Figure 5-12. The benefit is even more apparent when using periodic batch optimization and a robust cost function, as shown in Figure 5-13.

The blue and red lines are the pose graph (PG) and the green and cyan are the RPG. The error is computed by aligning to poses from the first session. The error given is then the average horizontal distance from the estimated pose to the corresponding groundtruth pose. Initially the error is high because there are no large loop closures, so the error comes from dead reckoning drift of the system. Slight angular error can easily generate large translation error when moving over a large area. After the initial loop closures are applied the error goes down sharply and then gradually decreases as more information is added.

These results show the benefit of using a robust cost function in comparison to the squared cost. When using squared cost a bad measurement can have a large impact on the solution and increase the error. As more measurements are added a bad measurement will eventually end up in the optimization. By continuously adding measurements these bad measurements can be detected by comparison with existing measurements between the same nodes.

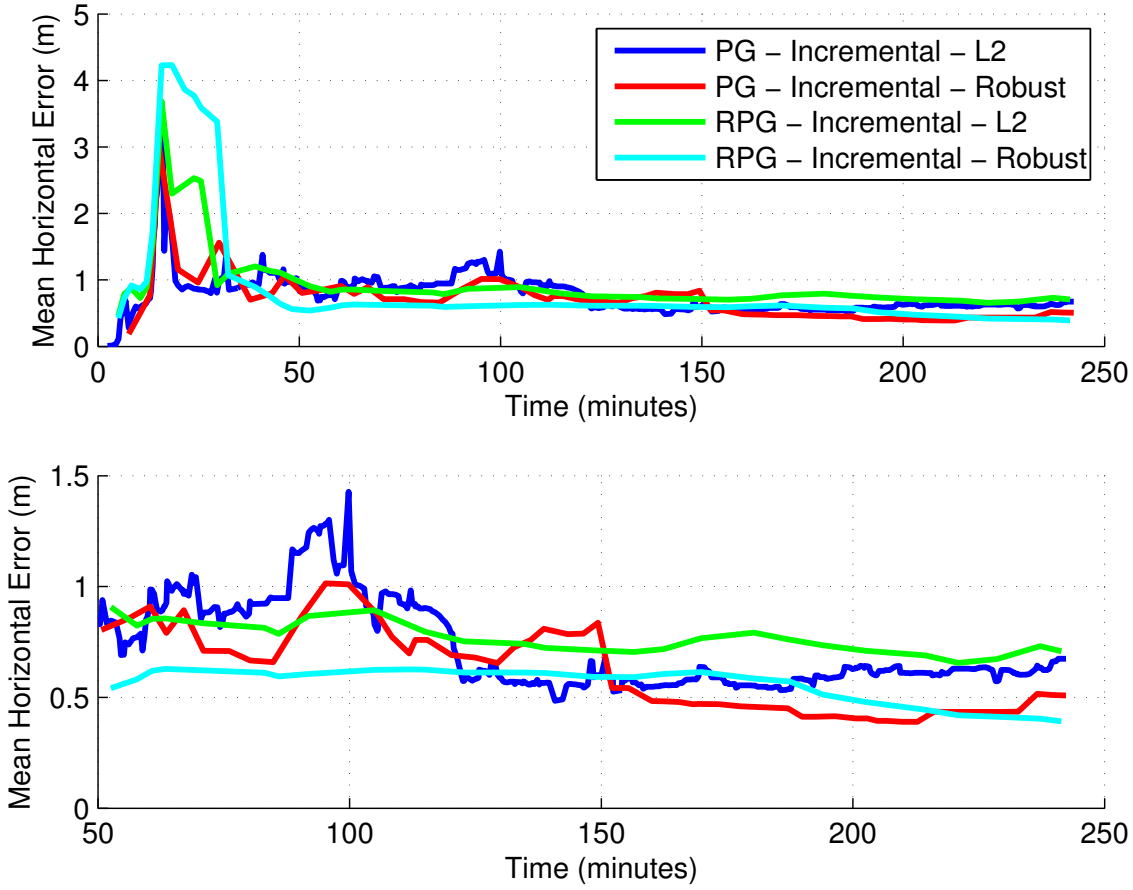


Figure 5-12: Estimation error for a subset of the poses over time using iSAM incrementally. The blue and red lines show the error for the full pose graph using squared and pseudo-Huber cost respectively. The green and cyan lines are the estimation errors for the reduced pose graph. The estimation error is computed as the Euclidean distance, in the XY plane, between the estimated position and the groundtruth.

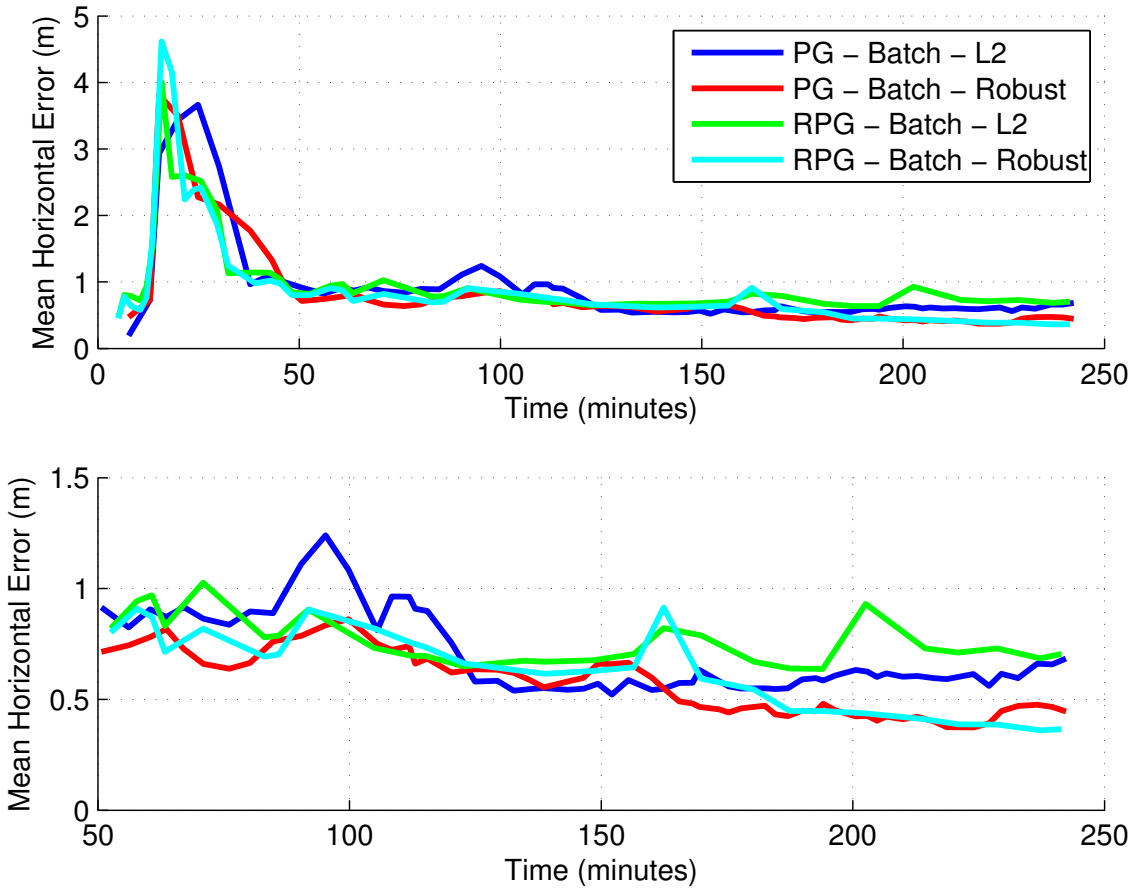


Figure 5-13: Estimation error for a subset of the poses over time using iSAM in batch mode. The blue and red lines show the error for the full pose graph using squared and pseudo-Huber cost respectively. The green and cyan lines are the estimation errors for the reduced pose graph. The estimation error is computed as the Euclidean distance, in the XY plane, between the estimated position and the groundtruth.

5.4.4 Complexity analysis

In this section we analyze the complexity of the reduced pose graph algorithm. We will look at the complexity of Algorithm 4 in terms of the number of poses N, measurements M, partitions K, and the size of the environment A.

The reduced pose graph mapping is run on a thread separate from the visual odometry. The sequential matches from the visual odometry thread are added to a queue. Each time the mapping thread enters the update method it processes all the matches in the incoming queue and finishes by updating the map. We assume that on each update there are constant number of frames in the queue. In the table below we provide the time complexity for each step.

Step	Lines	Complexity
Update transforms	2 - 5	$O(1)$
Extract descriptors	6	$O(1)$
Check local loop closure	7	$O(1)$
Check global loop closure	9	$O(N)$
Update active transform	12	$O(1)$
Add measurement	14 - 15	$O(1)$
Update map	20	$O(N^3 + M)$

Updating the transformation is constant time it only depends on the size of a single pose. Extracting the descriptors is linear in the number of features in a single frame. The local loop closure consists of a lookup for the nearest partition and then an alignment between the frames. The partitions are linearly indexed and stored in a vector so the lookup is constant. Only a single frame in a partition is tested by alignment. The alignment depends on the number of features in each frame so the operation is constant. The global loop closure needs to first search the appearance

based index which is $O(N)$ [8]. Adding a new node is a constant operation because the estimate is not updated immediately when a new node is added. The final step is the map update, this includes updating the estimate of the variables given the new measurements and the updating the space partitioning given these new estimates.

We use the iSAM library for computing the map estimate. The complexity of that algorithm is presented in [50]. The algorithm consists of incremental steps and a periodic batch update. For simplicity we only consider that batch step here. The complexity is very much dependent on the structure of the problem. For general problems the worst case is when the information matrix becomes fully dense and the complexity is $O(N^3 + M)$. For the SLAM problem the problem is typically very sparse, for pure exploration the complexity is $O(N + M)$, and when connectivity is restricted by limited sensor range and the graph can be reduced to a planar graph the complexity is $O(N^{1.5} + M)$.

The number of partitions grow with the area that the robot has traversed so it will be at most proportional to the total area, i.e. $K = O(A)$ where K are the number of partitions and A the area of the environment. The number of poses N in the graph are determined on line 17 in Algorithm 4. A pose is added each time a new place is explored and this should result in the number of poses to grow with the number of partitions, but because the current estimate of the robot is used to determine if the robot is in a new place or not it can fail if the robot is not properly localized to the map. This can result in unnecessary poses to be added. Let E be the number of times the, is in a new location test, fails. Then the number of poses grow as $O(A + E)$. If the test fails for some fraction of the time then the number of poses will still grow linearly with time even though at a significantly reduced rate. The effect of this can be seen in Figure 5-14. Even though the growth of the number of poses has significantly reduced there is still a slight increase in the number of poses

over the 4 hour period.

To guarantee a bound on the graph size it will eventually be necessary to remove the extraneous poses. A comparison of the reduced pose graph with and without node removal is shown in Figure 5-15. Here we use the node removal method used in the edge pruning algorithm we compared to earlier.

In the reduced pose graph the number of measurements will continuously grow. To bound the number of edges it is possible to combined edges that connect to the same pair of nodes. Though it might be beneficial to delay that reduction until multiple measurements are available. That way a robust cost could be applied when combining the edges. In the table below we compare the regular RPG algorithm with combining edges and removing nodes. We also consider removing nodes only at the end and then periodically for every 1000 measurements. The results suggest that we benefit from not removing nodes to early. Given how slow the reduced pose graph grows it would be sufficient to remove nodes only after several hours of operation. How to efficiently remove nodes and combine edges without sacrificing accuracy is an interesting topic of future research.

	Median	Mean	StdDev	Poses	Factors
RPG	0.34	0.35	0.16	1363	4333
RPG / combined edges	0.38	0.42	0.27	1364	3433
RPG / node removal at end	0.39	0.41	0.23	1003	2626
RPG / node removal every 1000	0.75	0.78	0.44	888	2458

The memory usage results from storage of keyframes, measurements, and state used for the map estimation. One keyframe is stored for each pose in the map estimate, so the storage for the keyframes is $O(N)$. The memory for the measurements is $O(M)$ because each measurement only relates to a constant number of poses. The

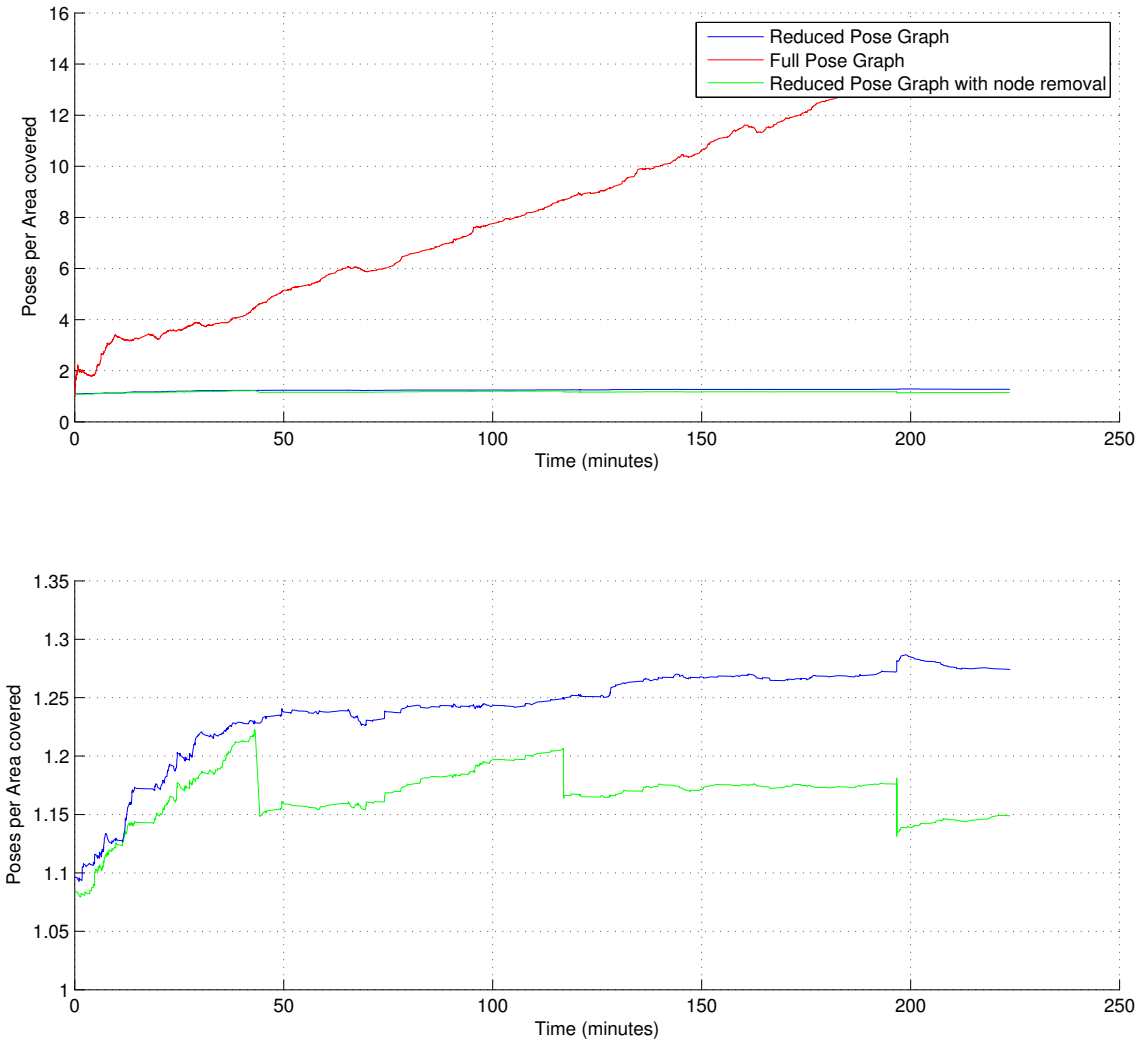
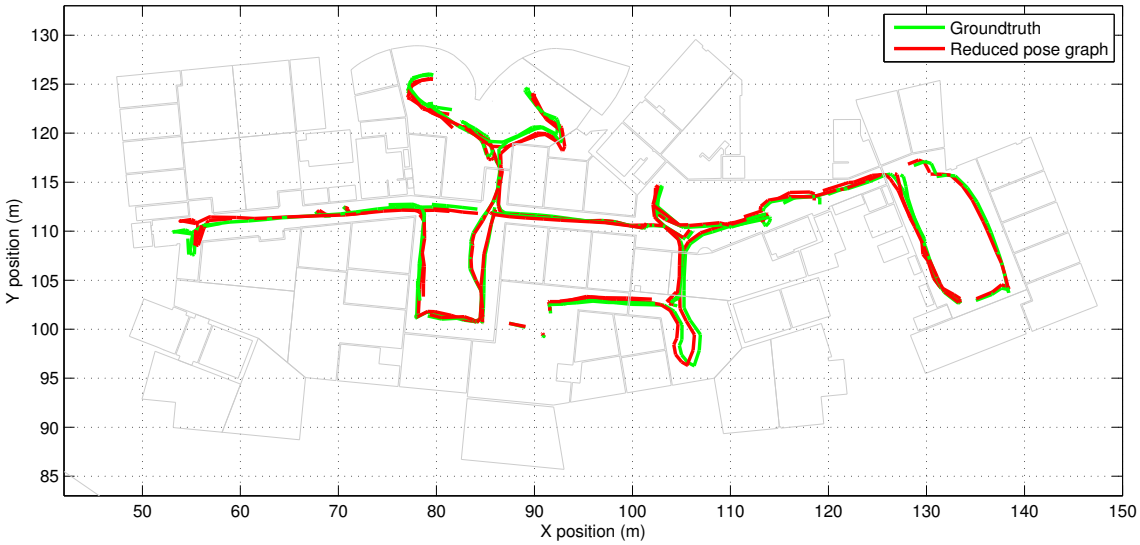
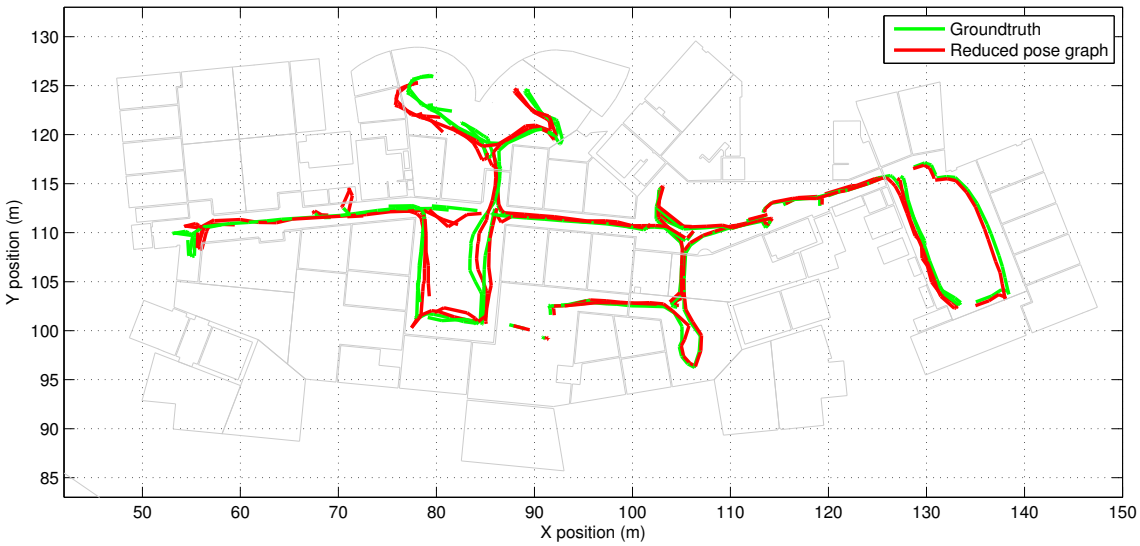


Figure 5-14: The number of poses as a ratio of area covered for the full pose graph and for the reduced pose graph with and without node removal.



(a) Reduced Pose Graph – 1363 poses



(b) Reduced Pose Graph with node removal – 888 poses

Figure 5-15: Comparison of a reduced pose graph with and without node removal.

map estimation maintains the square root information matrix, R, for the linearized non-linear least squares problem. The memory required depends on the density of R. It is $O(N)$ for exploration, $O(N \log N)$ for planar graphs, and $O(N^2)$ in general when the matrix is dense. Again, for the SLAM problem we typically have a very sparse matrix.

Even though the keyframes grow linearily there is a fairly high constant that needs to be considered. On average we store around 200 features per keyframe. For each feature we store the position of the feature with respect to the frame, the level it was detected on, a tracking id and a descriptor. When we use a 32 byte BRIEF descriptor, each frame takes around 10KB of memory. While the constants for the measurements and the R factor are in the 10's of bytes.

5.4.5 Long-term Multi-floor Mapping

We evaluated the system on nine hours of data that was collected in the Stata Center. The data was collected with the PR2 robot over a 6 month period. It includes visits to multiple floors in the building. An overview of the map constructed is shown in figure 5-17. The height has been exaggerated to make it easier to see each floor. The points are the features extracted from the stereo views associated with the poses in the map. As shown in Figure 5-16 the rate at which nodes are added to the graph reduces as time progresses. When new areas were first explored this rate increased as each new place was recorded. The loop proposal and the optimization modules grow in complexity as more nodes are added to the graph. However, these modules account for only a small fraction of the total running time of the system and are much reduced when compared to the traditional full pose graph. The majority of computation time is spent on frame registration and feature extraction, both of which

Duration of Experiment	6 months
Operation time	9 hours
Distance Traveled	11 km
VO keyframes	630K
Failed VO frames	87K
Registrations	303K
Loop proposals	30K

Table 5.1: Approximate figures of interest corresponding to the 10 floor experiment illustrated in Figure 5-17.

are constant time. Visual odometry runs at 30Hz on a separate thread, and loop closure runs at 2Hz.

5.5 Discussion

In this chapter we have described how we can reduce the complexity of the pose graph algorithm when operating for a long time duration in the same environment. As discussed in Section 5.4.4, the complexity of the algorithm depends on how well the robot is localized when traveling through previously mapped areas. An ideal situation would be a high quality camera navigating in a feature rich and well-lit environment. In this case we would expect the camera to stay well localized at all times. Failures would be more likely to occur in situations with few features, rapid camera motion, and/or poor lighting. In these situations, the tracking can be improved by tightly integrating IMU measurements with the motion estimate [76].

Another challenge is presented by dynamic environments. This can include objects moving in front of the camera or gradual changes occuring in the environment. The vision system has some robustness to objects that partially obscure the camera view because a set of inliers is found when the rigid body motion is computed. This

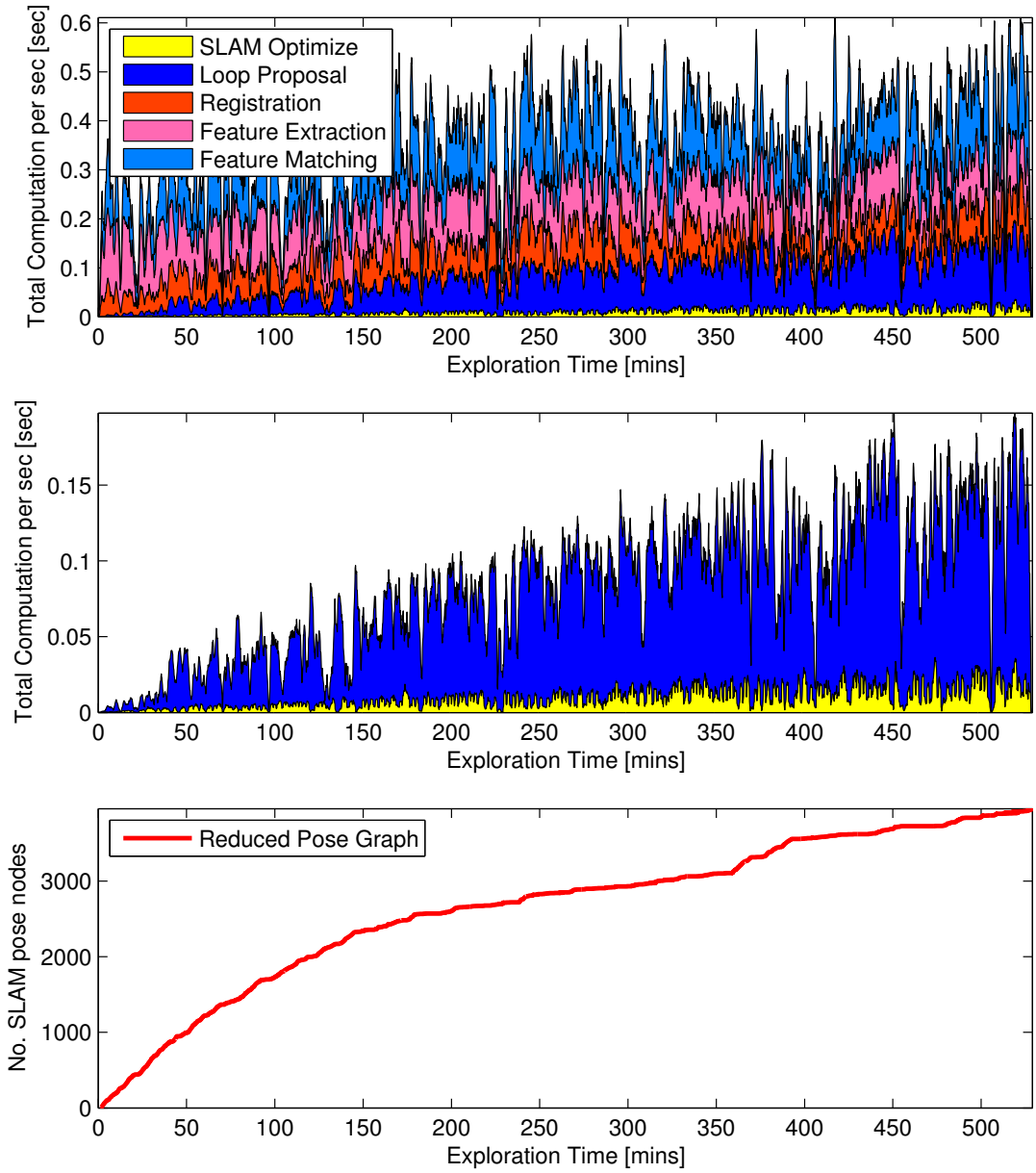


Figure 5-16: Timing results when using a **reduced pose graph** for a 9 hour sequence. The top plot shows the time of each component of the SLAM system as a function of exploration time. The middle plot shows the time for those components that have growing time complexity. The bottom plot shows the number of nodes in the graph.

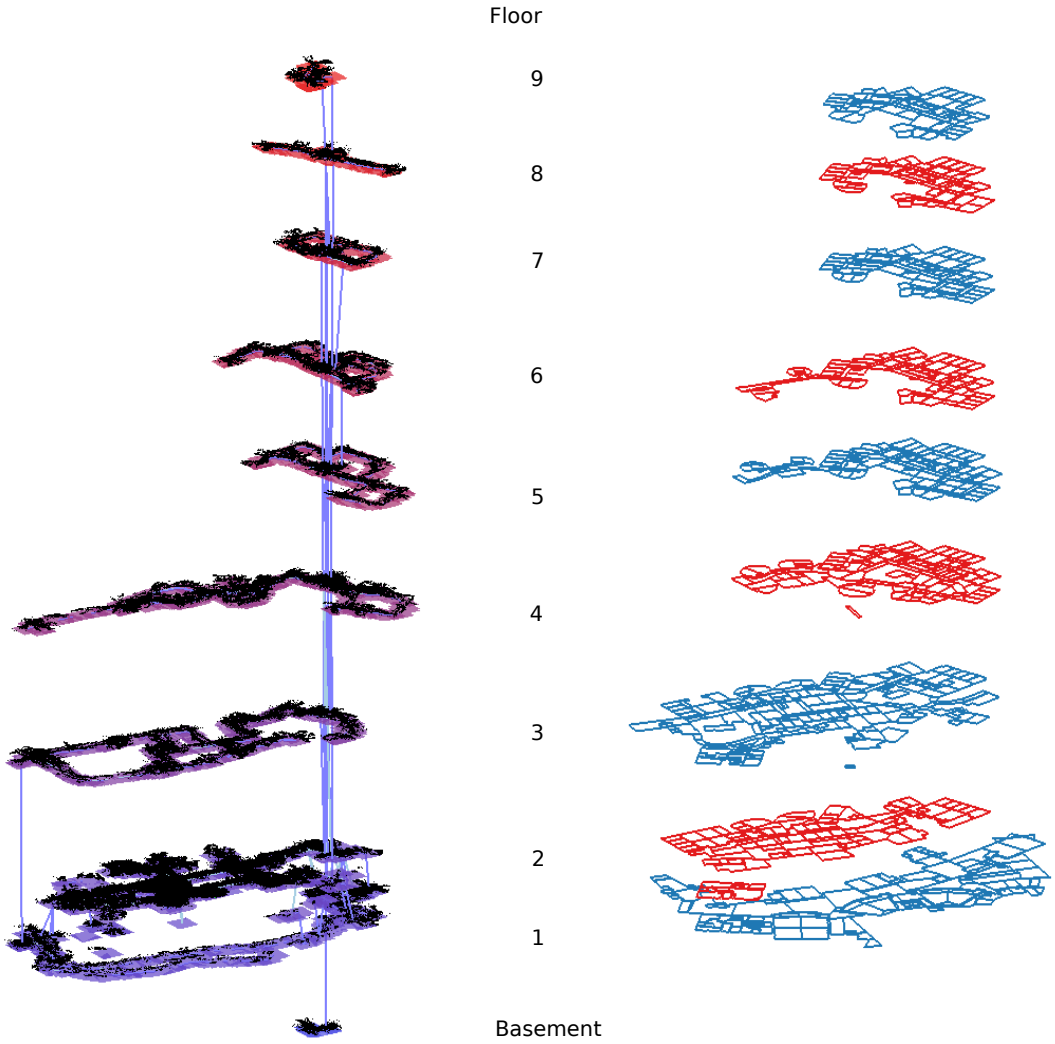


Figure 5-17: Left: A map of ten floors of the MIT Stata Center created using the reduced pose graph in combination with a real-time visual SLAM system. The data used was collected in 14 sessions spanning a six month period. The total operation time was nine hours and the distance traveled was 11km. Elevator transitions are shown as vertical blue lines. The view is orthographic and the vertical axis has been exaggerated to make it easier to see each floor. The 2nd floor is approximately 90m across. Right: Floor plans for each of the floors that were mapped.

results in features on moving objects to be rejected. On the other hand, if the object completely blocks the view of the camera, then the vision system will fail. We encountered this when traveling in elevators; in that case, we used IMU data to detect the vertical motion. This is of course a very restricted type of motion. Sensor fusion that could invalidate the camera motion estimate in a more general setting would be beneficial to the robustness of the system.

When operating for a long duration of time in the same environment we can expect gradual change. The system is robust to occasional objects appearing and disappearing, assuming there are enough background features that can be used for localization. On the other hand if a majority of the environment changes, then this would affect the localization accuracy. For example, we can consider a conference room where each time the robot visits the room, the configuration of the chairs is modified. Assuming that the chairs are needed to localize, the RPG algorithm would simply keep the representation from the first visit to the room. If the room changed significantly enough, then the robot would only perform dead-reckoning while in the room. If it spent enough time in the room, the position estimate would drift, and, as discussed before, the poor localization would result in new nodes to be added.

This problem could be addressed in several ways. One approach would be to always add nodes if the system fails to align to the existing map and then rely on node removal to bound the growth. This would cause the map to continuously change. A better approach would be to incorporate higher-level understanding in the representation, e.g. that chairs in a conference room tend to move around. In this case, one could perhaps use a two level mapping approach, by including a short-term transient map that can be "thrown away", while also keeping a stable background map. In this way the robot could localize in the room using the current configuration yet not try to remember it for later visits. A related approach was pursued by Biber

and Duckett [1], who maintained maps over multiple temporal scales for long-term dynamic mapping in two dimensions with lidar.

The results we have presented are from an indoor dataset which is typically a constrained environment. For this situation, the uniform partitioning of the space worked well. In more general situations containing large open spaces, it might be worth considering different partitioning strategies. For instance if distant features are commonly used for localizing it might be better to explicitly model these structures and then build constraints between them as the robot navigates around the environment. Also considering a non-uniform partitioning might be useful, e.g. a single room, or to represent some closed section of the environment with its own model. That approach would lead to a submapping strategy similar to what was used in the Atlas framework developed by Bosse et al. [5].

In Chapter 3 we presented methods for localization and mapping using sonars in an underwater environment. When operating underwater we often encounter environments that are have sparse features. In those situations it might be useful to group together a cluster of features and estimate their structure, similar to what we proposed above, and then to continually estimate the transformation between these structures. Another problem is that the natural structure of the environment is often repetitive, which could cause failures for loop closures based on a single view. To avoid this, it would be possible to combine multiple views and try to achieve loop closures that cover larger areas. Another thing to consider is that for underwater imaging we frequently need to carry our own light source, which can cast view dependent shadows, making matching more difficult. There have been, however several successful applications of vision for SLAM in the underwater domain [55, 65, 86] that give us confidence that the RPG algorithm would be successful in a wide range of underwater applications.

Chapter 6

Kinect Monte Carlo Localization

Once a model of an environment has been constructed, it can be useful to perform localization with respect to the prior map. In this chapter we describe how the output of the visual SLAM algorithm can be used to generate a simplified 3-D model of the environment and how it can be used for efficient Monte Carlo Localization using an RGB-D camera. [1]

6.1 Creating a 3-D Building Model

In many situation it can be useful to localize relative to a prior map. Using the pose estimates from the visual SLAM system, a 3D model of the environment can be constructed by reprojecting the points associated with each pose into a common reference frame. An example of such a model is illustrated in Figure 6-1 (top).

[1] The work presented in this chapter was performed in close collaboration with M. Fallon and has been previously published in [26]. The contribution of this thesis was the development of the GPU based scene prediction that enabled on-line operation of the system. Fallon led the development of the particle filter estimation algorithm and performance evaluation, which are included in this chapter to provide a complete description of the system. Fully connecting KMCL with the output of the reduced pose graph algorithm presented in Chapter 5 is an item for future research.

This representation provides a visually appealing reconstruction of the building and could be further processed to form a volumetric mesh reconstruction. Additionally a voxel-based occupancy tree could be used to aid path planning.

For the purposes of robust localization, we can also generate a simplified model of a large planar structure which is (1) unlikely to change (2) anchored on the pose graph and (3) of very small size (<10MB for the entire building). This approach extracts large planes from a single point cloud using RANSAC and then progressively grows the planes using nearby scans. The resulting plane-based model is presented in Figure 6-1 (bottom). This can be further extended by extracting the planes locally and then associating them with the poses, so that it is not necessary to perform the plane extraction process the second time around.

6.2 Monte Carlo Localization

Monte Carlo Localization (MCL) is a probabilistic localization algorithm that has been successfully used to localize a robot in a 2D occupancy grid using a laser ranger finder [16, 101], while we will use a 3D planar world model and a RGB-D camera to localize. It localizes the robot by estimating the posterior distribution of the vehicle location given all its sensor measurements.

Let x_t be the robot location at time t and without loss of generality we assume that the robot makes a single sensor measurement z_t at each time step. In addition it receives a control input u_t at each time step. The set of all measurements up to time step t is given by Z_t. It is assumed that the motion model is Markov and that the sensor measurements are independent, given the position of the robot. The graphical model for the distribution is shown in Figure 6-2, and the joint probability

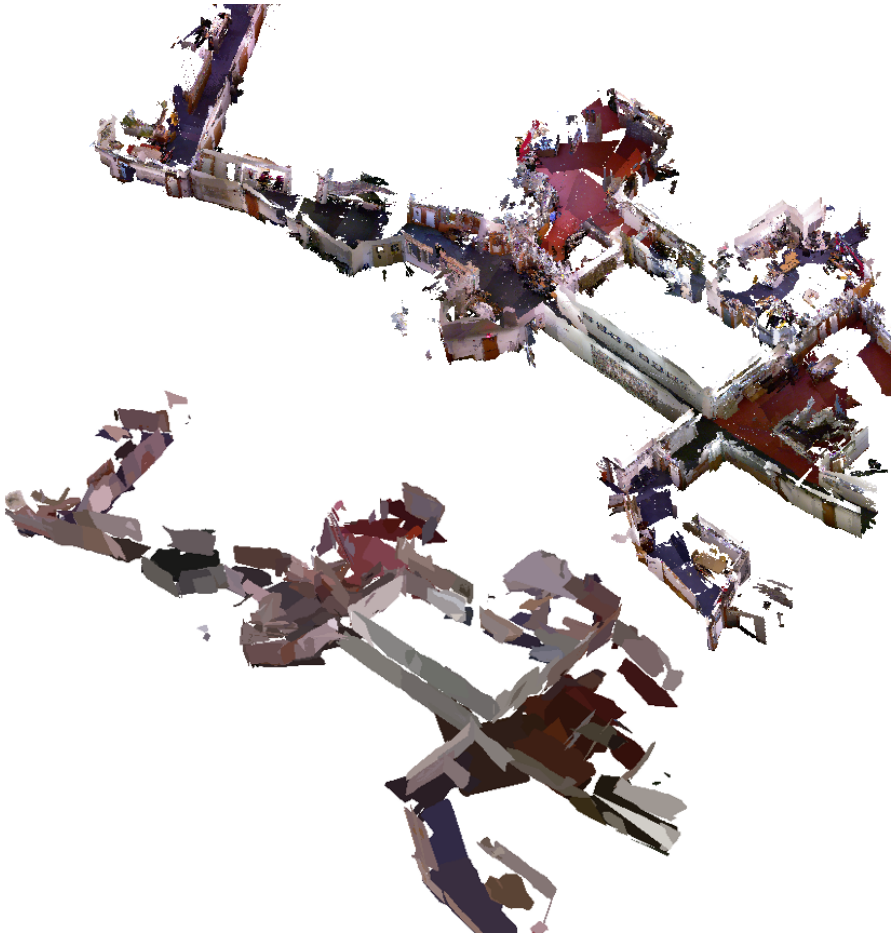


Figure 6-1: Output of the Visual SLAM system: A dense 3-D point cloud of the 2nd floor of the Stata Center (top). Input to the KMCL Localization system: A simplified model made up of only large planar objects (bottom).

density for the model is given by:

$$p(X_t, Z_t, U_t) = p(x_0) \prod_{i=0}^{t} p(x_i|x_{i-1}, u_i) p(z_i|x_i) p(u_i) \tag{6.1}$$

To localize the robot we are interested in the current position x_t given all measurements and controls. This is provided by the posterior probability density $p(x_t|Z_t)$,

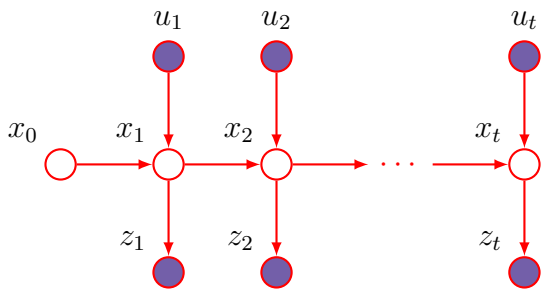


Figure 6-2: Graphical model for localization.

which can be computed using a Bayesian filter approach [99].

The Bayes filter allows us to update the posterior recursively:

$$p(x_t|Z_t, U_t) = \frac{p(z_t|X_t, Z_{t-1}, U_t)}{p(z_t|Z_t, U_t)} p(x_t|Z_{t-1}, U_t) \tag{6.2}$$

$$\propto p(z_t|X_t) \int p(x_t|X_{t-1}, U_t) p(x_{t-1}|Z_{t-1}, U_t) \mathrm{d}x_{t-1} \tag{6.3}$$

$$= p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) p(x_{t-1}|Z_{t-1}, U_{t-1}) \mathrm{d}x_{t-1} \tag{6.4}$$

where in the last step we use the assumption of conditional independence of measurements given robot trajectory and independent controls. The motion model is given by $p(x_t|x_{t-1}, u_t)$, and $p(z_t|x_t)$ is the sensor model or the likelihood of the sensor measurement. The bottom left image in Figure 6-3 gives an example of a sensor measurement (z_t) from a RGB-D camera, and top right is the predicted image based on the map.

In general it is not tractable to compute the full posterior distribution. To track the posterior distribution we use a particle filter as was initially proposed for robot localization in [16]. More generally, particle filtering was initially proposed by Gordon et al. [34] also known as Sequential Monte Carlo (SMC).

A particle filter represents the posterior distribution by a set of weighted Monte

Carlo importance samples. At each time step the particles are propagated according to Equation 6.4. Given a set of N particles where x_t^k is particle k at time step t they are first propagated using the motion model $p(x_t|x_{t-1}, u_t)$. The propagation is done by random sampling from the motion model. Then for each particle the sensor model, $p(z_t|x_t^k)$, is evaluated and the weights are updated accordingly. In the next section we will describe a method to efficiently evaluate the sensor model, which is crucial to achieving good performance.

6.3 Likelihood Function

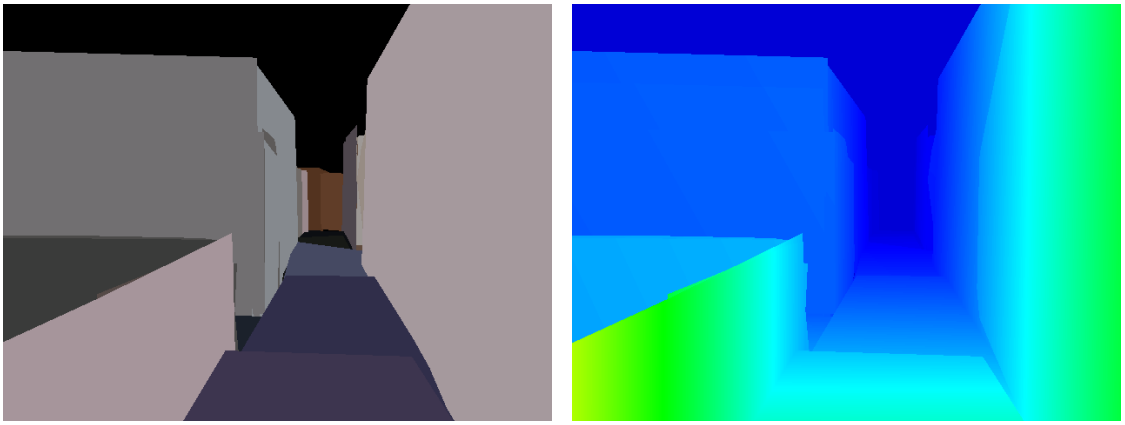
We now turn our attention to how to efficiently compute the likelihood for each particle using the current depth image provided by the camera. At each time step the particle weights are updated using the likelihoods. The likelihoods are computed for each particle. Efficiency of the computation impacts how many particles can be used and thus the accuracy of the filter.

The computational complexity is significantly reduced by down sampling the incoming RGB-D image by a factor of 16–32. From our experiments we have seen that from the 640x480 pixel image/cloud, a sufficiently informative likelihood function is possible using a 20x15 image. The likelihood is then evaluated on this smaller image and indicates how likely each particle is, given its position, the current sensor measurement, and the map.

In the following sections we describe and compare two different likelihood functions. First, we describe a novel approach which simulates model views and then compares them to the measured data. Then, in Section 6.3.2 a method similar to the ICP scoring function is described. Finally, a short comparison of the two methods is given in Section 6.3.2.

6.3.1 Generated Range-Image Ray-to-Plane Function

We propose to compute the likelihood of a particle pose by directly generating the range image that would have been measured from that location using the prior 3-D model. This range image is generated by finding the closest intersection between a



(a) Simulated Color Image (b) Simulated Depth Image

(c) Measured Color Image (d) Measured Depth Image

Figure 6-3: Using a prior 3-D building model we efficiently generate simulated depth and color views (top) which are then compared to RGB-D camera data (bottom). Using a particle filter, hundreds of similar views are evaluated and used to estimate the sensor pose in 3-D.

ray through the optical center and a point on the image plane with the prior map for each point in the simulated range image. However instead of brute-force ray-tracing to find the intersection of each ray with each plane, it is possible to render each plane directly to the image plane instead. The distance to the closest plane can then be queried from the associated Z-buffer. The Z-buffer is used by computer graphics systems to decide, for each pixel, which object is closest to the virtual camera and hence should be rendered — essentially being the depth of the nearest object. This rendering approach is supported in all modern Graphical Processing Units (GPU) and the depth image for each particle can be rendered very efficiently in this manner.

Our implementation uses the OpenGL library to render the simulated image for each particle. This is similar to the approach in [81] which used gradients in the color image, however here we use the depth information. When OpenGL renders an image, such as our 3-D model, it natively uses the Z-buffer to determine hidden surface removal. After rendering, the Z-buffer values can be read and used to compute the likelihood of the current sensor measurement conditioned on the pose of the particle.

GPU implementation

Before comparing the sensor depth with the Z-buffer values there are a few technical issues in the rendering pipeline that need to be taken into account. First all plane polygon vertices are transformed, using the model transform, into the camera coordinate frame. (Note that OpenGL defines the camera look axis along the negative Z-axis.)

After the model transformation, the projection transform is applied. This projection creates the clip coordinates, which are in homogeneous coordinates, with each rendering primitive clipped to the box $(-w, -w, -w), (w, w, w)$. The z values

are projected in such a way that everything outside the range $[-z_n, -z_f]$ is clipped, where z_n and z_f are the near and far range of the z-axis [110]. For the Kinect RGB-D sensor z_n is 0.7m and z_f is 20m. Finally the projected inverse depth in the range $[1/z_n, 1/z_f]$ is mapped to $[0, 1]$ before it is written to the Z-buffer which we can then access.

For a camera calibration matrix K

$$K = \begin{bmatrix} f_x & 0 & -c_x & 0 \\ 0 & f_y & -c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

the projection matrix can be written as KP

$$P = \begin{bmatrix} 2/W & 0 & 1 & 0 \\ 0 & 2/H & 1 & 0 \\ 0 & -\frac{z_f+z_n}{z_f-z_n} & -\frac{2z_f z_n}{z_f-z_n} & \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{6.6}$$

where W and H are the image width and height respectively. An example depth image is illustrated in Figure 6-3. Next we compare it to the measured image (also illustrated) to produce a likelihood for the particle pose.

To achieve good performance when using the GPU there are several things to consider. It is essential to try to minimize the amount of information being sent between the GPU and the CPU. To address that we used vertex buffers to transfer the geometry onto the GPU. Thus only a single invocation is required for each particle. We also tested moving the computation of the likelihood function onto the GPU.

This was implemented using a custom GLSL (OpenGL Shading Language) shader. The likelihood function was precomputed on the CPU and sent to the GPU as a texture. Finally, we implemented an aggregated sum method to add together the individual point values for each particle. Thus it is sufficient to send an image that is of size $N \times N$ for N^2 particles instead of all the depth values.

We compared having the likelihood computation and the aggregated sum on the GPU vs the CPU. The depth image was still generated using the GPU. The model used contained 5024 triangles and covered the 2nd floor of the MIT Stata Center. The results are shown in Table 6.1. Each result is the average of 100 runs, and we compared for different image sizes. We found that when using the smaller sizes there is little benefit in having the GPU do the likelihood computation. This is because for each particle a draw call is issued to the GPU, which incurs some overhead. In the results reported later, the CPU version was used for the likelihood computation and sum aggregation.

A further improvement to the algorithm would be to use geometric instancing and viewport arrays (which have recently been added to OpenGL). Thus the particles could be sent in groups and fewer calls made. For very large models a further improvement might be to store the models in an efficient structure like octrees and only render the parts that are visible. However, one needs to be careful that the work for each particle to determine visibility does not outweigh simply rendering that part of the model, as the GPU is very efficient at culling and clipping vertices that are not visible.

Image size	Time on CPU (ms)	Time on GPU (ms)
20 x 15	3.8	3.7
40 x 30	4.4	4.1
80 x 60	7.2	5.1
160 x 120	11.6	6

Table 6.1: Performance of the likelihood computation using 10x10 particles.

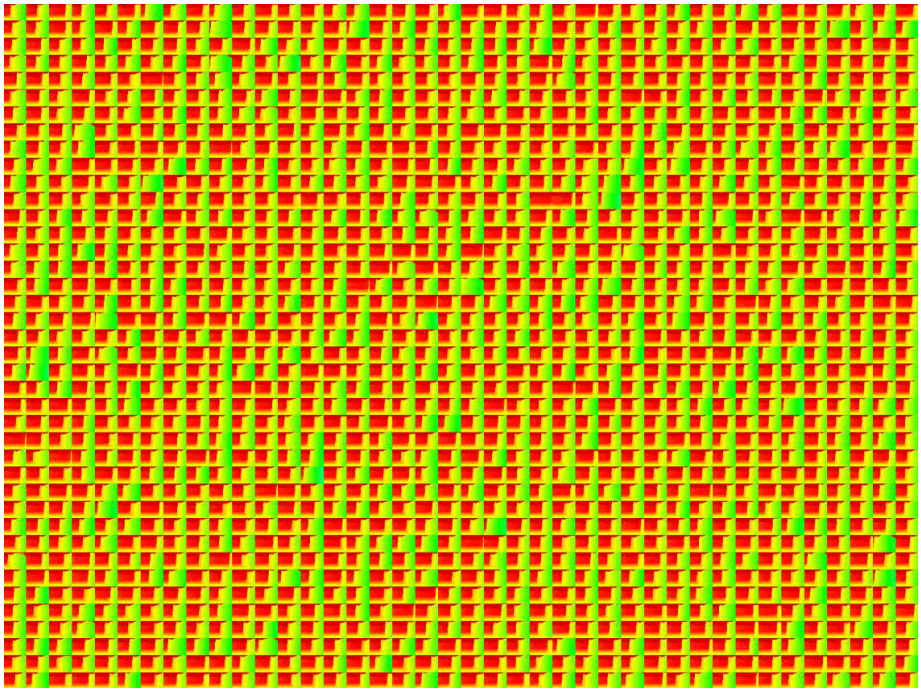


Figure 6-4: An example of the contents of the depth buffer for multiple particles.

Likelihood Formulation

To compute the particle likelihood either the inverse depth Z-buffer values can be converted to depth, or the RGB-D depth data can be converted to the inverse depth. As the accuracy of camera depth measurements is a linear function of inverse depth we propose to formulate the likelihood function in inverse depth space. This is similar to the approach taken in monocular camera SLAM, for example [11].

For a given particle $x_t^{(k)}$ at time step t, the generated inverse depth image, $Z_{(k)}^G = (z_{(k)}^0, \ldots, z_{(k)}^N)$, containing N points is generated as described above. For each inverse depth pixel, z_t^i, in the measured image $Z_t^M = (z_t^0, \ldots, z_t^N)$, the likelihood is evaluated as follows

$$p(z_t^i | x_t^{(k)}) = \beta c_r \mathcal{N}(z_t^i; z_{(k)}^i, \sigma_d^2) + (1 - \beta)\mathcal{U}(0, 1) \tag{6.7}$$

where the inverse depth varies in the range $(0, 1)$. An appropriate normalization constant, c_r, was added for the truncated normal distribution and the inverse depth variance σ_d^2 was chosen to be $0.1m^{-1}$. The addition of the uniform distribution supports heavy-tailed behavior, and in doing so each point in the cloud has only a small effect on the overall likelihood function. The parameter $\beta = 0.01$ was found to give good experimental performance.

The overall likelihood of the particle is then the product of the point likelihoods across the entire cloud

$$p(\mathbf{Z}_k | x_t^{(k)}) = \prod_{i=1}^{N_i} p(z_t^i | x_t^{(k)}) \tag{6.8}$$

The procedure is repeated for each of the particles in the cloud and the weights are then updated to produce an estimate of the posterior distribution at the current time

$$\tilde{w}_t^{(k)} \propto \tilde{w}_{t-1}^{(k)} p(\mathbf{Z}_k | x_t^{(k)}) \tag{6.9}$$

Residual resampling is carried out whenever the effective sample size of the particle set falls below 0.5.

Figure 6-5 illustrates the inverse depth parametrization of the likelihood function described in Equation 6.7. It has been projected onto the depth axis, evaluated for a series of nominal model depths. For points at shorter range the function is much more discriminative than for points at larger ranges — directly matching the

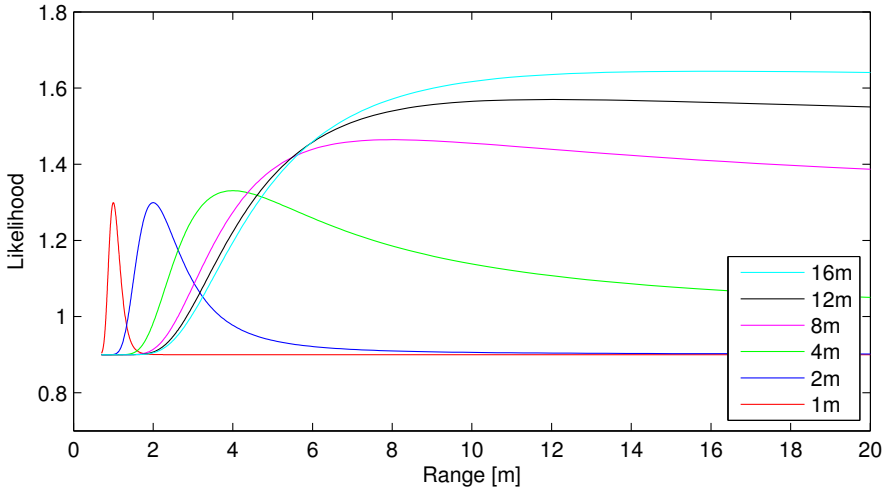


Figure 6-5: Inverse Depth-parametrized likelihood function evaluated for a variety of nominal model depths. This parametrization evaluates high-range depth points with higher variance than shorter ranges — accurately representing the underlying disparity measurement uncertainty.

measurement error distribution. This allows us to take advantage of Kinect depth measurements all the away up to 20 meters away.

While this approach is a principled way of utilizing the noisy long-range Kinect RGB-D data, discretization of the measurement depth values (in the range $(0-2047)$) as well as texture dependent biases are evident at long ranges (>15m).

In addition to depth information, this method can be supplemented with basic color information by reading the color buffer to produce colored model views similar to that illustrated in Figure 6-3. By defining a color-based distance metric, individual color pixels could contribute to the particle weighting using an extra term in Equation 6.7. This would be useful in long shapeless corridors for example.

6.3.2 Pure Euclidean Point-to-Plane Function

For comparison to the method proposed above, we also developed a more traditional likelihood function. It operates in a manner somewhat similar to the *cost function* of the Iterative Closest Point (ICP) by evaluating the Euclidean distance between the RGB-D point cloud (transformed by the proposed particle pose) and the planar submap.

For a given particle $x_t^{(k)}$, the RGB-D cloud is first transformed onto the particle pose and the minimum point-to-plane distance is found by comparing each point, $z_{(k)}^i$, from the cloud to each plane, s_j, in the submap mentioned above

$$d_{i,\min}^{(k)} = \arg\min_j \|z_{(k)}^i - s_j\| \tag{6.10}$$

where $\|*\|$ represents the distance from point to plane.

Given this distance, the individual point likelihood is then evaluated using a form similar to Equation 6.7, with the per particle likelihood being the product of the individual point likelihoods.

Comparison Between Functions

Both of these methods demonstrate similar accuracy when there is a good alignment between the measurements and the building model (See Section 6.4). While the latter method has been demonstrated to be significantly more computationally intensive, additionally it demonstrates a failure mode that our proposed method does not suffer from. ICP search is well known to suffer from poor convergence when incorrectly initialized and the Euclidean-based particle filter has been observed to demonstrate a similar behavior. An example situation is that presented in Figure 6-3. The method

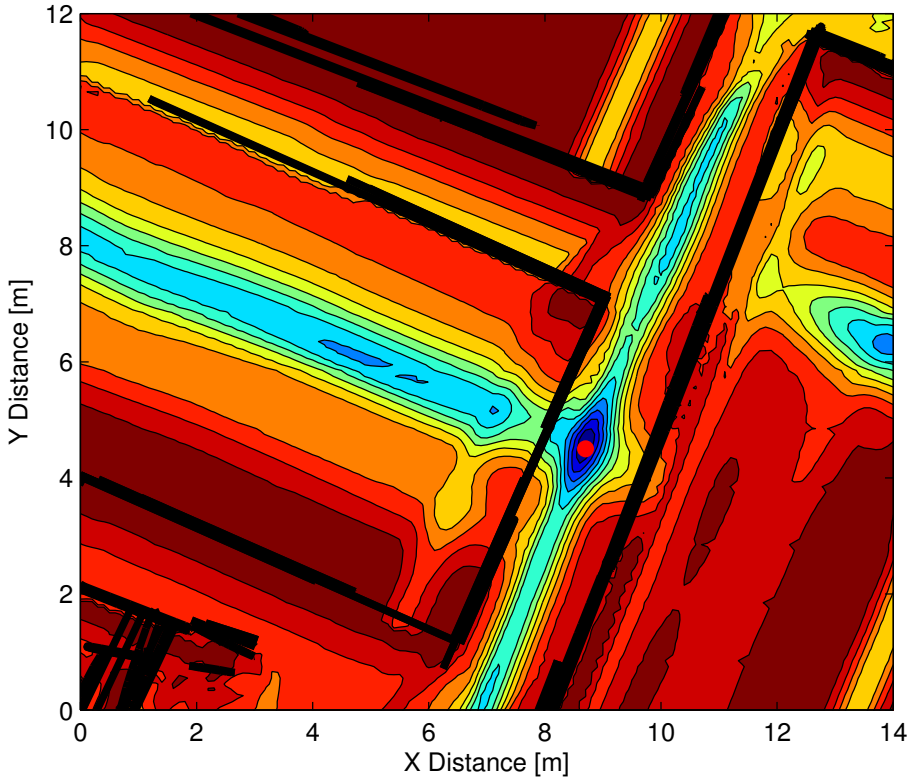


Figure 6-6: The generative likelihood function evaluated as a contour surface with 10 cm spacing (in 2D) around the location illustrated in Figure 6-3 (denoted by the red circle). The walls and railings are in black. The multi-modal surface has a broad peak at the correct location ensuring stable MCL.

is prone to incorrectly associating depth points from the wall to the model's railing, resulting in an incorrect local minimum and a possible failure of the particle filter.

The likelihood surface for the proposed generative method does not suffer from this issue as illustrated by the generative likelihood surface illustrated in Figure 6-6. The ICP surface, by comparison, would have significant peak 1 m behind the true location by mismatching the railings in the model to the lower part of the wall.

6.4 Results

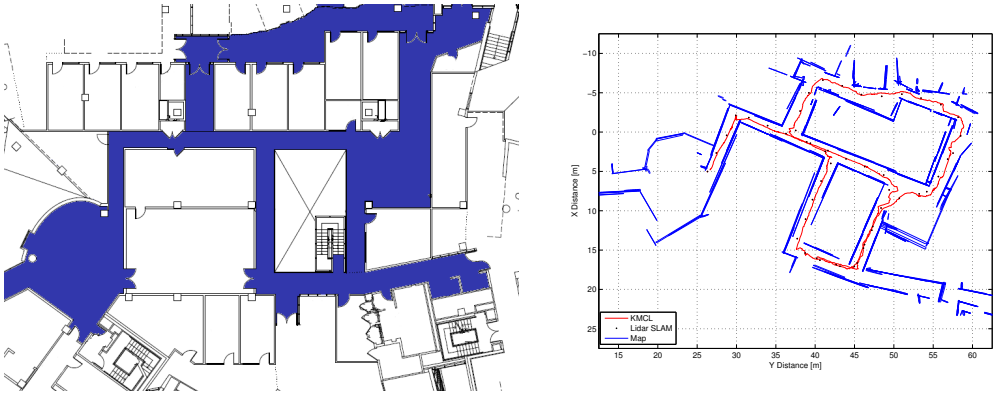
The accuracy and the performance of the system was evaluated using datasets from several different platforms: a robotic wheelchair, a Willow Garage PR2, an Ascending Technologies quadrotor, a man-portable mapping device and an RWI B21 wheeled mobile robot.

Each platform was equipped with a forward facing Microsoft Kinect camera. For all the platforms except the quadrotor, an accurate estimate of the ground truth pose was estimated using a laser base pose graph SLAM and a Hokuyo UTM-30LX laser rangefinder mounted in the primary plane of motion, and was used to estimate the localization error of the KMCL system. The height of the sensor varied between 1–2 meters, which demonstrates the flexibility of this approach.

6.4.1 Localization accuracy

Each dataset was post-processed using the KMCL algorithm and a trajectory generated. Figure 6-7 shows a trajectory for the PR2 dataset — the red lines are the vehicle trajectory and the black dots are the ground truth estimated using the LIDAR. In Table 6.2 numerical results are provided using 350 particles.

No major failures of the localization algorithm occurred in these experiments (i.e. the entire particle set diverging), although troublesome locations containing little or no visual or geometric features do exist within the building - such as long corridors or blank walls. In these locations, the particle set naturally disperses somewhat until some conclusive geometric information is observed, at which point the particle distribution coalesces to a single mode and continues accurate tracking. These situations, despite being representative of the underlying position uncertainty, result in an increased error result in Table 6.2 and Section 6.4.2. For small particle sets these



(a) Floor Plan (b) Willow Garage PR2

Figure 6-7: Top-down view of performance for the PR2 localizing. The main walls in the map are shown in blue while each robot's Kinect MCL output is shown with a red line. Black dots indicate the ground truth position during the runs (estimated using LIDAR SLAM) which indicates accurate localization.

situations can also result in particle filter divergence (as discussed in Section 6.4.2).

The error metrics we chose were median absolute error, and the percentage of SLAM poses lying within a 3σ interval of the weighted particle mean. As indicated in the table, typical median error is of the order of 40 cm. This value is inflated due to poor performance in the aforementioned locations. While Table 6.2 seems to indicate increased error with increased speed, we believe that improved VO integration will reduce this effect.

In particular the data for the man-portable system exhibits significant motion blur and frequent visual odometry failure; thus the results with this system indicate the robust nature of our approach. Additionally the quality of the 3-D map and the choice of image downsample factor are other parameters that could be studied so as to improve performance further.

Platform	Duration	Distance	Speed	Median Error	3σ %
Units	seconds	m	m/s	m	%
Man-carried	94	99	1.05	0.66	52
Wheelchair	180	215	1.20	0.48	75
Quadrotor	45	$\sim$30	$\sim$0.66	n/a	n/a
PR2	266	126	0.47	0.30	0.90
B21	349	152	0.43	0.33	0.84

Table 6.2: Performance of the KMCL algorithm (using 350 particles) for the trajectories in Figure 6-7.

6.4.2 Performance

The performance of the algorithm was quantified with a series of Monte Carlo runs using the robotic wheel chair dataset. The dataset consisted of a 3 minute sequence covering 215 meters. The number of particles used was varied between 12 to 400. For each setting the algorithm was run for 20 independent runs.

To compute the accuracy of each run the particle filter trajectory was aligned with the LIDAR poses. As shown in Figure 6-8, the results show accuracy and the stability of the particle filter for varying numbers of particles.

Comparing the two different likelihood functions, we observed roughly equivalent localization accuracy with equal numbers of particles — with the Euclidean likelihood function being slightly more accurate. The main difference between the two methods was in the computation time. For 100 particles and a frame rate of 10 Hz, the generative method is real-time, while the Euclidean method is 5 times slower than real-time. The slow pace of the Euclidean likelihood precluded us from testing with 200 and 400 particles (where the gap was even wider).

In conclusion a stable real-time operation with 350 particles was realized, using a 4-core 2.53GHz Pentium Core2 powered laptop with an Nvidia Quadro 1700M with

32 Cores. The computation was split between two CPU cores, one for the data capture and VO, and another for the Monte Carlo localization, processing 7–8 frames per second. In regions of low uncertainty as few as 10 particles are required for operation. Implementation of an adaptively resizing particle cloud would be useful in such circumstances [32]. Further optimizations such as submapping and instanced rendering will allow more particles which improves accuracy, thus improving operations in challenging locations. Finally, incorporating IMU will improve the particle propagation, filling in gaps where the VO fails, in addition to giving absolute measurement of vertical axis with respect to gravity.

In this chapter we demonstrated a system for localization using RGB-D cameras. Because these cameras are fairly inexpensive compared to laser range finders this technology is interesting for enabling localization on a wide range of robots. Future improvements that could be made to the system include: (1) incorporating IMU information, and (2) increasing the efficiency of the scene prediction algorithm. In addition, it would be interesting to combine the technique presented in this chapter with the visual SLAM algorithm presented in Chapters 5 and 4, which could improve the robustness of the whole system.

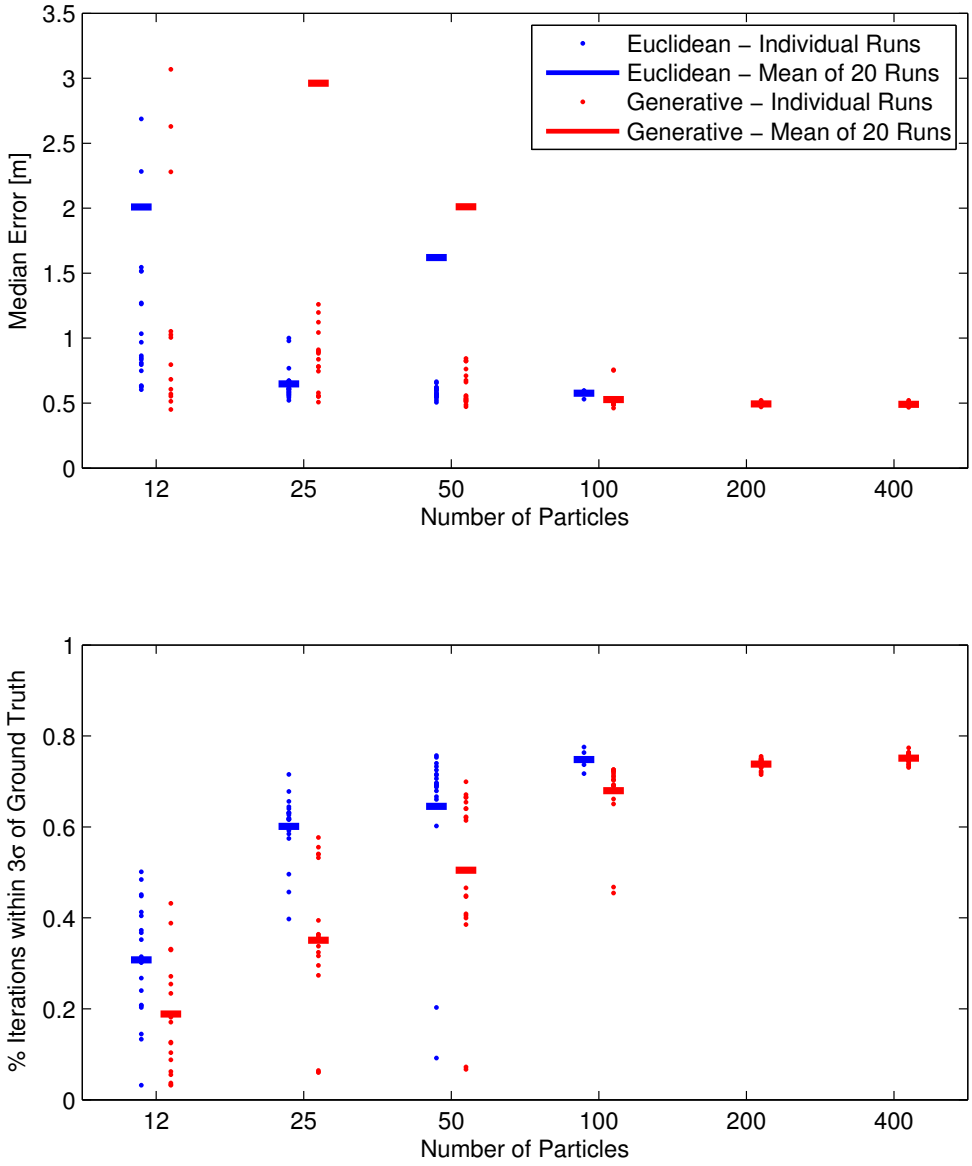


Figure 6-8: Performance metrics for both likelihood functions (averaged for 20 separate runs). Typical performance for 100 particles is of the order of 0.5 m median error and 78% of estimates within 3σ of the true location. Note that for some failed runs with low particle numbers the median error is greater than 3.5m.

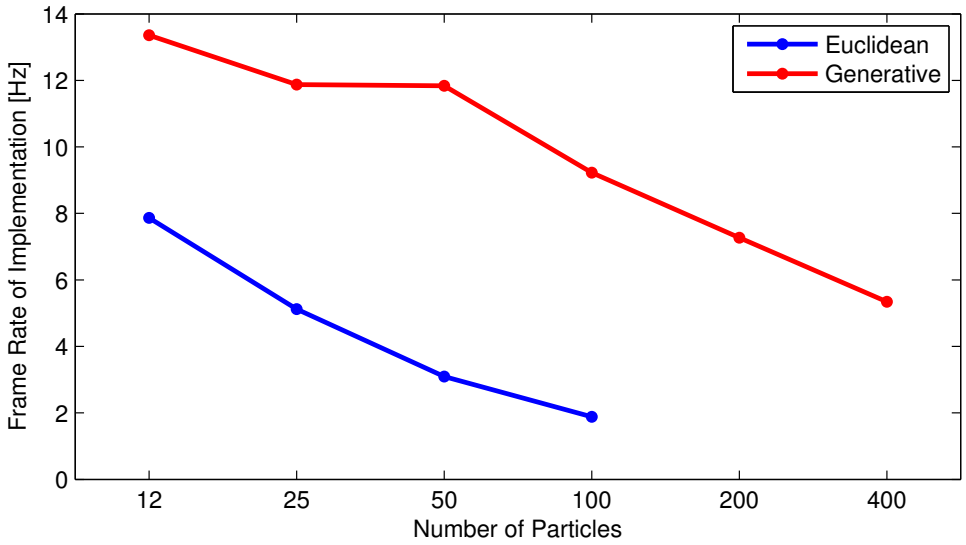


Figure 6-9: Timing statistics for both likelihood functions. In the case of the later, real time operation (at 10Hz) has been achieved with 100 particles and the frequency reduces linearly with the number of particles.

Chapter 7

Conclusion

One of the fundamental problems of mobile robotics is *simultaneous localization and mapping*. In this thesis we have presented imaging-sonar aided navigation for underwater applications and a complete visual SLAM solution which was evaluated on an indoor dataset. To address challenges in mapping imposed by lifelong autonomy we proposed a reduced pose graph (RPG) model that proved robust while drastically reducing the complexity of the optimization. The effectiveness of the RPG was evaluated using an extensive dataset that included nine hours of data, traveling eleven kilometers, and collected over a six month period.

7.1 Contributions

This thesis has made several contributions to localization and mapping both for underwater navigation — demonstrating an on-line application — and a vision SLAM system supporting lifelong operations. Following are the main contributions of the thesis:

1. *We described an imaging-sonar based SLAM system that was applied to autonomous ship hull inspection.*

   When operating underwater a sonar system is often the preferred sensor. We extended our previous work on imaging-sonar aided navigation to support full 6DoF motion, fusion of IMU and pressure measurements, and removing the assumption of a horizontal sea-floor. Also, we developed a distributed estimation architecture that enabled the inclusion of camera constraints [54] in addition to the sonar measurements. Finally, the system was demonstrated in on-line experiments enabling accurate target re-acquisition using the improved hull navigation.

2. *We presented a multi-session feature based SLAM system for an underwater vehicle equipped with a forward looking sonar.*

   We presented a method for SLAM using sparse features for underwater navigation. In addition we proposed a global re-alignment method that was then applied to multi-session mapping. Then, we looked at map merging and how complexity could be managed by only augmenting the map when new features were observed. Finally, robustness was provided by maintaining multiple map representations through each session.

3. *We developed a complete pose graph based visual SLAM system.* In addition to stereo-vision and RGB-D, information from sensors such as a robot's wheel odometry and IMU can also be incorporated and result in improved robustness in situations where the vision-only system would fail. We also showed how elevator transits can be detected — enabling seamless multi-floor mapping.

4. *To manage the time complexity of the formulation we proposed and evaluated*

a reduced pose graph model.

The graphical representation of the SLAM problem has many advantages, including: retaining sparseness so it is efficient to optimize, individual measurements are kept so it is possible to recover from errors, and the measurements can be re-linearized as the estimate improves. The downside is that the complexity grows as more variables are added over time. In contrast, the RPG maintains a minimal representation by partitioning the world. In addition, the RPG continuously integrates new information as constraints on the existing nodes and the accuracy of the map improves over time. Finally, the model was evaluated on an extensive vision dataset collected in the MIT Stata Center.

5. *Finally, we presented an efficient GPU based scene prediction algorithm that was applied to localization using an RGB-D camera.*

   Motivated by the recent availability of inexpensive RGB-D cameras we explored their use in localization as a replacement for planar LIDARs which have been successfully used for localization. In addition the camera is not restricted to planar motions. By developing an efficient GPU scene prediction algorithm, it was possible to evaluate the likelihood functions for enough particles to enable accurate localization.

7.2 Future Work

Even though much progress has been made in solving SLAM, deploying robotics systems in the field that rely on SLAM for extended duration still eludes us. The mobile robots that currently operate over long durations typically perform localization using a prior map, or rely on an external infrastructure to localize.

The key challenge is robustness. Because in SLAM the map is continuously updated, one needs to guarantee that errors do not permanently corrupt the map. Also, the computational complexity must stay within the capability of the robot. Furthermore, understanding the effects of the feedback from the mapping process to the robot controller when operating on-line is an important topic.

In most applications we are interested in having the maps as accurate as possible. The accuracy of the estimation will always depend on the model used, and how well its assumptions and approximations capture the underlying true distribution. In the work presented here we have assumed corrupted Gaussian distributions and independence of measurements. These error distributions are functions of the sensors, environment, motion, and the algorithm used. Analyzing these distributions to come up with better approximations would improve the accuracy and robustness of the overall estimation.

For the best performance the RPG algorithm relies on good tracking, though on occasion it might loose track of the map, causing it to add unnecessary poses to the map. This can be resolved by improving the tracking, and by editing the graph to remove nodes that are not needed.

For underwater applications it would be interesting to explore applying some of the approaches that have been developed in the vision community to the sonar imagery, e.g. building descriptors for features in sonar images, and developing a vocabulary of acoustic words. In the ship hull inspection work the sonar and image constraints were computed independently of each other. An improvement would be the ability to obtain correspondences between a sonar and a camera view. The FBN data shows an example of a target seen at a distance with the sonar, which then passed directly under the vehicle to be captured on the camera. A constraint between those two views would provide valuable information in the state estimation.

For the SLAM problem in general, there are still many challenges. Even though the work presented in this thesis is considered large scale mapping, the MIT Stata Center only covers 700,000 sq.ft. The MIT campus consists of 12 million square feet of floor space, and is only a small part of Cambridge. A typical MIT graduate student manages to find their way around Cambridge quite effectively, even though they might make a wrong turn here or there.

The map representation is an important factor for scalability. It must be possible to store and query the map efficiently, as well as computing the estimation itself. The representation also affects the robustness of the system and how change is treated. Looking for a high level description of the environment is interesting here — considering work from object recognition and visual semantics. In the underwater domain we might handle features extracted from bio-fouling on the ship hull differently then the water intake or weld lines. In a dynamic environment a description that is too detailed will be less useful for localization, as the detailed description is more likely to change, e.g. the cup might no longer be there, or the chair may now be pushed back.

When operating over a very long time, there will be gradual changes in the environment caused by the change of seasons — the tree in winter is the same tree in summer, but looks differently. Again, the description will have a big impact on how these changes are treated. Similarly, underwater, a storm will change the sand ripples, and possibly shift some objects around. Of course in some situations these changes might actually be the subject of interest. But fully coping with change is a challenging unresolved problem, in part because it is hard to resolve between a change in the environment and inaccuracy in the robot's position estimate.

Perceptual ambiguity is another problem to consider. In our work, combining appearance based recognition with geometric-verification proved sufficient to re-localize

the robot if lost, or at the start of a new session. As the robots work in larger areas, it is unclear if perceptual ambiguity will become more of a problem.

Complex and dynamic environments provide a challenge where other objects or vehicles block a large portion of the current view, forcing the camera to measure motion relative to the object in view, which may disagree with other sensors like the IMU and the DVL (depending on where it is pointing). Fusing these inconsistent measurements must be avoided. This requires the system to have an improved world model that can explain these differences. Currently, when using vision, robustness to moving objects is typically achieved using inlier detection, however that would fail if that static world were not visible. A good example is being aboard a ship, where the consequences of the mixed signals from our eyes and our equilibrium sensors often have ill consequences — though typically we adjust after a while.

In conclusion, we see there are many challenges that need to be addressed before we achieve a fully autonomous system that can reliably navigate in large scale, complex, dynamic environments over extended period of time. Improved representations, higher level understanding of the environment, fusion of multiple sensor modalities, and improved recovery from errors are some of the capabilities that will be required to meet these challenges.

Bibliography

[1] P. Biber and T. Duckett. Dynamic maps for long-term operation of mobile service robots. In *Robotics: Science and Systems (RSS)*, page 17, 2005.

[2] P. Biber and T. Duckett. Experimental analysis of sample-based maps for long-term SLAM. *Intl. J. of Robotics Research*, 28(1):20–33, 2009.

[3] P. Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 2743–2748, October 2003.

[4] R. Bloss. Mobile hospital robots cure numerous logistic needs. *Industrial Robot: An International Journal*, 38(6):567–571, 2011.

[5] M. Bosse, P. Newman, J. Leonard, and S. Teller. An atlas framework for scalable mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1899–1906, 2003.

[6] M. Bosse, P. Newman, J. Leonard, and S. Teller. Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework. *Intl. J. of Robotics Research*, 23(12):1113–1139, December 2004.

[7] M.C. Bosse. *ATLAS: A framework for large scale automated mapping and localization*. PhD thesis, Massachusetts Institute of Technology, 2004.

[8] C. Cadena, D. Gálvez, F. Ramos, J.D. Tardós, and J. Neira. Robust place recognition with stereo cameras. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010.

[9] C. Cadena, J. McDonald, J. Leonard, and J. Neira. Place recognition using near and far visual information. In *Proceedings of the 18th IFAC World Congress*, August 2011.

[10] R.O. Castle, G. Klein, and D.W. Murray. Wide-area augmented reality using camera tracking and mapping in multiple regions. *Computer Vision and Image Understanding*, 115(6):854 – 867, 2011.

[11] Laura A. Clemente, Andrew J. Davison, Ian Reid, José Neira, and Juan D. Tardós. Mapping large loops with a single hand-held camera. In *Robotics: Science and Systems (RSS)*, June 2007.

[12] M. Cummins and P. Newman. Highly scalable appearance-only SLAM - FAB-MAP 2.0. In *Robotics: Science and Systems (RSS)*, Seattle, USA, June 2009.

[13] A. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, June 2007.

[14] F. Dellaert. Square Root SAM: Simultaneous location and mapping via square root information smoothing. In *Robotics: Science and Systems (RSS)*, Cambridge, MA, 2005.

[15] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *Intl. J. of Robotics Research*, 25(12):1181–1203, December 2006.

[16] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte Carlo localization for mobile robots. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 1999.

[17] E. Eade, P. Fong, and M.E. Munich. Monocular graph SLAM with complexity reduction. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3017–3024, October 2010.

[18] B. Englot, H. Johannsson, and F. Hover. Perception, stability analysis, and motion planning for autonomous ship hull inspection. In *Proceedings of the International Symposium on Unmanned Untethered Submersible Technology (UUST)*, 2009.

[19] C. Estrada, J. Neira, and J.D. Tardós. Hierarchical SLAM: Real-time accurate mapping of large environments. *IEEE Trans. Robotics*, 21(4):588–596, 2005.

[20] R. Eustice. *Large-Area Visually Augmented Navigation for Autonomous Underwater Vehicles*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2005.

[21] R. Eustice, H. Singh, J. Leonard, M. Walter, and R. Ballard. Visually navigating the RMS Titanic with SLAM information filters. In *Robotics: Science and Systems (RSS)*, June 2005.

[22] R.M. Eustice, H. Singh, and J.J. Leonard. Exactly sparse delayed-state filters for view-based SLAM. *IEEE Trans. Robotics*, 22(6):1100–1114, December 2006.

[23] N. Fairfield, A. G. Kantor, and Wettergreen D. Real-time SLAM with octree evidence grids for exploration in underwater tunnels. *J. of Field Robotics*, 2007.

[24] M. F. Fallon, J. Folkesson, H. McClelland, and J. J. Leonard. Relocating underwater features autonomously using sonar-based SLAM. *IEEE J. Ocean Engineering*, 2012. To Appear.

[25] M. F. Fallon, H. Johannsson, J. Brookshire, S. Teller, and J. J. Leonard. Sensor fusion for flexible human-portable building-scale mapping. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Algarve, Portugal, 2012.

[26] M.F. Fallon, H. Johannsson, and J.J. Leonard. Efficient scene simulation for robust Monte Carlo localization using an RGB-D camera. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1663–1670, St. Paul, MN, May 2012.

[27] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[28] J. Folkesson and H.I. Christensen. Graphical SLAM - a self-correcting map. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 383–390, 2004.

[29] J. Folkesson, J. Leedekerken, R. Williams, and J. Leonard. Feature tracking for underwater navigation using sonar. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, San Diego, CA, October 2007.

[30] J. Folkesson, J. Leederkerken, R. Williams, A. Patrikalakis, and J. Leonard. A feature based navigation system for an autonomous underwater robot. In *Field and Service Robotics*, volume 42, pages 105–114, 2008.

[31] J. Folkesson and J. Leonard. Autonomy through SLAM for an underwater robot. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, 2009.

[32] Dieter Fox. Adapting the sample size in particle filters through KLD-sampling. *Intl. J. of Robotics Research*, 22(12):985–1003, December 2003.

[33] G. Gawarkiewicz, J. Nelson, R. He, RW Fulweiler, J. Goff, T. Grothues, and E. LaBrecque. Shelf/slope processes: Science opportunities and issues relating to the ooi pioneer array, 2011.

[34] N.J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings F on Radar and Signal Processing*, volume 140, pages 107–113, 1993.

[35] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2D and 3D mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Anchorage, Alaska, May 2010.

[36] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems (RSS)*, June 2007.

[37] J. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 1, pages 318–325, 1999.

[38] T. Hafting, M. Fyhn, S. Molden, M.B. Moser, and E.I. Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806, 2005.

[39] S.E. Harris and E.V. Slate. Lamp Ray: Ship hull assessment for value, safety and readiness. In *OCEANS '99 MTS/IEEE. Riding the Crest into the 21st Century*, volume 1, pages 493–500, 1999.

[40] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003. Second Edition.

[41] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, Zaragoza, Spain, 2010.

[42] F.S. Hover, R.M. Eustice, A. Kim, B.J. Englot, H. Johannsson, M. Kaess, and J.J. Leonard. Advanced perception, navigation and planning for autonomous in-water ship hull inspection. *Intl. J. of Robotics Research*, 31(12):1445–1464, October 2012.

[43] A. Huang, E. Olson, and D. Moore. LCM: Lightweight communications and marshalling. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010.

[44] A.S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an RGB-D camera. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, Flagstaff, USA, August 2011.

[45] V. Ila, J.M. Porta, and J. Andrade-Cetto. Information-based compact pose SLAM. *Robotics, IEEE Transactions on*, 26(1):78–93, February 2010.

[46] H. Johannsson, M. Kaess, B. Englot, F. Hover, and J.J. Leonard. Imaging sonar-aided navigation for autonomous underwater harbor surveillance. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010.

[47] H. Johannsson, M. Kaess, M.F. Fallon, and J.J. Leonard. Temporally scalable visual SLAM using a reduced pose graph. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013. To appear.

[48] S.J. Julier and J.K. Uhlmann. A counter example to the theory of simultaneous localization and map building. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 4238–4243, 2001.

[49] S.J. Julier, J.K. Uhlmann, I. Ind, and MO Jefferson City. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[50] M. Kaess. *Incremental Smoothing and Mapping.* Ph.d., Georgia Institute of Technology, December 2008.

[51] M. Kaess, V. Ila, R. Roberts, and F. Dellaert. The Bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Intl. Workshop on the Algorithmic Foundations of Robotics, WAFR*, Singapore, December 2010.

[52] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J.J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[53] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. Robotics*, 24(6):1365–1378, December 2008.

[54] A. Kim. *Active Visual SLAM with Exploration for Autonomous Underwater Navigation*. PhD thesis, The University of Michigan, 2012.

[55] A. Kim and R.M. Eustice. Real-time visual SLAM for autonomous underwater hull inspection using visual saliency. *IEEE Trans. Robotics*, 2013. To appear.

[56] James C. Kinsey, Ryan M. Eustice, and Louis L. Whitcomb. A survey of underwater vehicle navigation: Recent advances and new challenges. In *IFAC Conference of Manoeuvering and Control of Marine Craft*, Lisbon, Portugal, September 2006. Invited paper.

[57] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, pages 225–234, Nara, Japan, November 2007.

[58] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 802–815, Berlin, Heidelberg, 2008. Springer-Verlag.

[59] O. Koch and S. Teller. Body-relative navigation guidance using uncalibrated cameras. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1242–1249. IEEE, 2009.

[60] K. Konolige and M. Agrawal. FrameSLAM: From bundle adjustment to real-time visual mapping. *IEEE Trans. Robotics*, 24(5):1066–1077, October 2008.

[61] K. Konolige and J. Bowman. Towards lifelong visual maps. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1156 –1163, October 2009.

[62] K. Konolige and J. Bowman. Towards lifelong visual maps. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1156–1163, October 2009.

[63] H. Kretzschmar, C. Stachniss, and G. Grisetti. Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, September 2011.

[64] F.R. Kschischang, B.J. Frey, and H-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2), February 2001.

[65] C.G. Kunz. *Autonomous underwater vehicle navigation and mapping in dynamic, unstructured environments*. PhD thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution, 2012.

[66] J. J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proc. IEEE Int. Workshop on Intelligent Robots and Systems*, pages 1442–1447, Osaka, Japan, 1991.

[67] Jongwoo Lim, Jan-Michael Frahm, and Marc Pollefeys. Online environment mapping using metric-topological maps. *Intl. J. of Robotics Research*, 31(12):1394–1408, 2012.

[68] F. Lu and E. Milios. Globally consistent range scan alignment for environmental mapping. *Autonomous Robots*, 4:333–349, April 1997.

[69] A. Mallios, P. Ridao, E. Hernandez, D. Ribas, F. Maurelli, and Y. Petillot. Pose-based SLAM with probabilistic scan matching algorithm using a mechanical scanned imaging sonar. In *OCEANS 2009-EUROPE, 2009. OCEANS '09.*, pages 1–6, May 2009.

[70] J. McDonald, M. Kaess, C. Cadena, J. Neira, and J.J. Leonard. 6-DOF multisession visual SLAM using anchor nodes. In *European Conference on Mobile Robotics*, Örbero, Sweden, September 2011.

[71] Wim Meeussen, Eitan Marder-Eppstein, Kevin Watts, and Brian P. Gerkey. Long term autonomy in office environments. In *ICRA 2011 Workshop on Long-term Autonomy*, Shanghai, China, 05/2011 2011. IEEE, IEEE.

[72] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid. RSLAM: A system for large-scale mapping in constant-time using stereo. *International Journal of Computer Vision*, pages 1–17, 2010. 10.1007/s11263-010-0361-7.

[73] M.J. Milford and G.F. Wyeth. Persistent navigation and mapping using a biologically inspired SLAM system. *Intl. J. of Robotics Research*, 29(9):1131–1153, August 2010.

[74] D.A. Mindell. Precision navigation and remote sensing for underwater archaeology. *Remote sensing in archaeology*, page 499, 2007.

[75] M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping

that provably converges. In *Intl. Joint Conf. on Artificial Intelligence*, pages 1151–1156. Morgan Kaufmann Publishers Inc., 2003.

[76] A.I. Mourikis, N. Trawny, S.I. Roumeliotis, A.E. Johnson, A. Ansar, and L. Matthies. Vision-aided inertial navigation for spacecraft entry, descent, and landing. *Robotics, IEEE Transactions on*, 25(2):264–280, 2009.

[77] P. Newman, G. Sibley, M. Smith, M. Cummins, A. Harrison, C. Mei, I. Posner, R. Shade, D. Schroter, L. Murphy, W. Churchill, D. Cole, and I. Reid. Navigating, recognising and describing urban spaces with vision and laser. *Intl. J. of Robotics Research*, 28, October 2009.

[78] K. Ni, D. Steedly, and F. Dellaert. Tectonic SAM: Exact, out-of-core, submap-based SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1678–1685, April 2007.

[79] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, volume 1, pages 652–659, June 2004.

[80] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, volume 2, pages 2161–2168, 2006.

[81] S. Nuske, J. Roberts, D. Prasser, and G. Wyeth. Experiments in visual localisation around underwater structures. In Andrew Howard, Karl Iagnemma, and Alonzo Kelly, editors, *Field and Service Robotics*, volume 62 of *Springer Tracts in Advanced Robotics*, pages 295–304. Springer Berlin / Heidelberg, 2010.

[82] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2262–2269, May 2006.

[83] R. Panish and M. Taylor. Achieving high navigation accuracy using inertial navigation systems in autonomous underwater vehicles. In *OCEANS, 2011 IEEE - Spain*, pages 1 –7, june 2011.

[84] P. Pinies, L.M. Paz, D. Galvez-Lopez, and J.D. Tardos. CI-Graph SLAM for 3D reconstruction of large and complex environments using a multicamera system. *J. of Field Robotics*, 27(5):561–586, September 2010.

[85] K. Pirker, M. Ruether, and H. Bischof. CD SLAM - continuous localization and mapping in a dynamic world. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, September 2011.

[86] O. Pizarro, R.M. Eustice, and H. Singh. Large area 3-d reconstructions from underwater optical surveys. *Oceanic Engineering, IEEE Journal of*, 34(2):150 –169, april 2009.

[87] D. Ribas, P. Ridao, J. Neira, and J.D. Tardós. SLAM using an imaging sonar for partially structured underwater environments. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[88] C. Roman and H. Singh. A self-consistent bathymetric mapping algorithm. *J. of Field Robotics*, 24(1):23–50, 2007.

[89] D.M. Rosen, M. Kaess, and J.J. Leonard. An incremental trust-region method for robust online sparse least-squares estimation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1262–1269, St. Paul, MN, May 2012.

[90] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Eur. Conf. on Computer Vision (ECCV)*, pages 430–443, Berlin, Heidelberg, 2006. Springer-Verlag.

[91] G. Sibley, C. Mei, I. Reid, and P. Newman. Adaptive relative bundle adjustment. In *Robotics: Science and Systems (RSS)*, Seattle, USA, June 2009.

[92] G. Sibley, C. Mei, I. Reid, and P. Newman. Planes, trains and automobiles – autonomy for the modern robot. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 285–292. IEEE, 2010.

[93] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Intl. Conf. on Computer Vision (ICCV)*, volume 2, page 1470, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[94] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.

[95] C. Stachniss and W. Burgard. Mobile robot mapping and localization in non-static environments. In *AAAI Conf. on Artificial Intelligence*, pages 1324–1329. AAAI Press, 2005.

[96] C. Stachniss and H. Kretzschmar. Pose graph compression for laser-based SLAM. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, 2011.

[97] S. Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous robots*, 15(2):111–127, 2003.

[98] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, AB Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *Intl. J. of Robotics Research*, 19(11):972–999, 2000.

[99] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, MA, 2005.

[100] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *Intl. J. of Robotics Research*, 23(7), 2004.

[101] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128, May 2001.

[102] J. Vaganay, L. Gurfinkel, M. Elkins, D. Jankins, and K. Shurn. Hovering autonomous underwater vehicle — system design improvements and performance evaluation results. In *Proc. Int. Symp. Unmanned Untethered Subm. Tech.*, Durham, NH, 2009.

[103] A. Walcott. *Long-Term Autonomous Navigation and Exploration in Dynamic Environments*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2011.

[104] M. Walter, F. Hover, and J. Leonard. SLAM for ship hull inspection using exactly sparse extended information filters. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1463–1470, May 2008.

[105] M.R. Walter. *Sparse Bayesian information filters for localization and mapping*. PhD thesis, Massachusetts Institute of Technology, 2008.

[106] M.R. Walter, R.M. Eustice, and J.J. Leonard. Exactly sparse extended information filters for feature-based SLAM. *Intl. J. of Robotics Research*, 26(4):335–359, 2007.

[107] Z. Wang, S. Huang, and G. Dissanayake. D-SLAM: A decoupled solution to simultaneous localization and mapping. *Intl. J. of Robotics Research*, 26(2):187–204, 2007.

[108] L. Whitcomb, D. Yoerger, and H. Singh. Combined Doppler/LBL based navigation of underwater vehicles. In *Proceedings of the International Symposium on Unmanned Untethered Submersible Technology (UUST)*, May 1999.

[109] L. Whitcomb, D. Yoerger, H. Singh, and D. Mindell. Towards precision robotic maneuvering, survey, and manipulation in unstructured undersea environments. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, volume 8, pages 45–54, 1998.

[110] R. Wright, B. Lipchak, and N. Haemel. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley Professional, fourth edition, 2007.

[111] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010.

[112] B. Yamauchi and R. Beer. Spatial learning for navigation in dynamic environments. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(3):496–505, June 1996.

[113] D. Yoerger, M. Jakuba, A. Bradley, and B. Bingham. Techniques for deep sea near bottom survey using an autonomous underwater vehicle. *Intl. J. of Robotics Research*, pages 416–429, January 2007.

[114] D.R. Yoerger and D.A. Mindell. Precise navigation and control of an ROV at 2200 meters depth. In *Proceedings of Intervention/ROV*, volume 92, 1992.